EstiNet Protocol Module 開發手冊

Release Date: May 2, 2018

Produced and maintained by EstiNet Technologies Inc.

閱讀本手冊建議：

已經有使用過 ESTINET 的 GUI 操作並知道如何執行模擬
已經取得 SOURCE CODE
對 C++不陌生
對網路概論有基本的認識

目標讀者：

- 想要在 ESTINET 上開發自己的模組
- 想要修改 ESTINET 上既有的模組

本手冊內容：

本手冊共分為 9 章，前兩章著重運用 GUI 的一些開發元件，讓開發者在開發自己的模組時更為便利；後面七章則是著重使用內部模組中常用的 API，以及常用的資料結構。

目　錄

第一章：**MDF 與 PROTOCOL STACK** **4**

第二章：**MDF 與 RUN TIME QUERY** **34**

第三章：模組間互相分享資料 **54**

第四章：**RUN TIME MESSAGE** **61**

第五章：不需要使用 **GUI** 的模擬執行方式 **66**

第六章：**TIMER** **71**

第七章：**EVENT** **82**

第八章：**PACKET** **88**

第九章：其它 **API** 練習 **99**

附錄 **105**

第一章：MDF 與 Protocol Stack

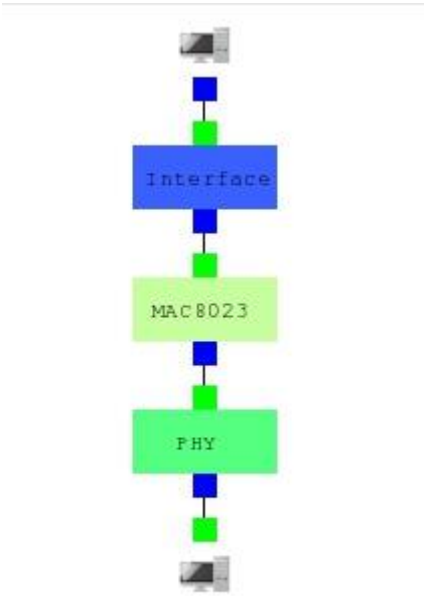
本章重點：
(1)、什麼是 MDF？什麼是 Protocol Stack？
(2)、怎麼改 MDF 及 Protocol Stack？
(3)、模擬引擎(estinetse)中的 API--VBIND、get_nid()
(4)、怎麼編繹 Source Coce？

下載練習：


Chapter1.tar.bz2

在 EstiNet 的模擬引擎中，提供了一個模擬的平台，內含有許多模組(Module)，每個模組都有不同的機制，像模擬有線 IEEE 802.3 的模組，無線的 IEEE 802.11 系列的模組等等。

許多模組所串接起來是一個網路設備的 Protocol Stack，下圖所示為 Host1 的 Protocol Stack。(註：每種網路設備的 Protocol Stack 不盡相同)



每個模組中都有一些設定值開放給使用者設定，因此需要一個介面以及描述模組的檔案，提供給 GUI 介面開放給使用者設定，這個檔案稱為 MDF。

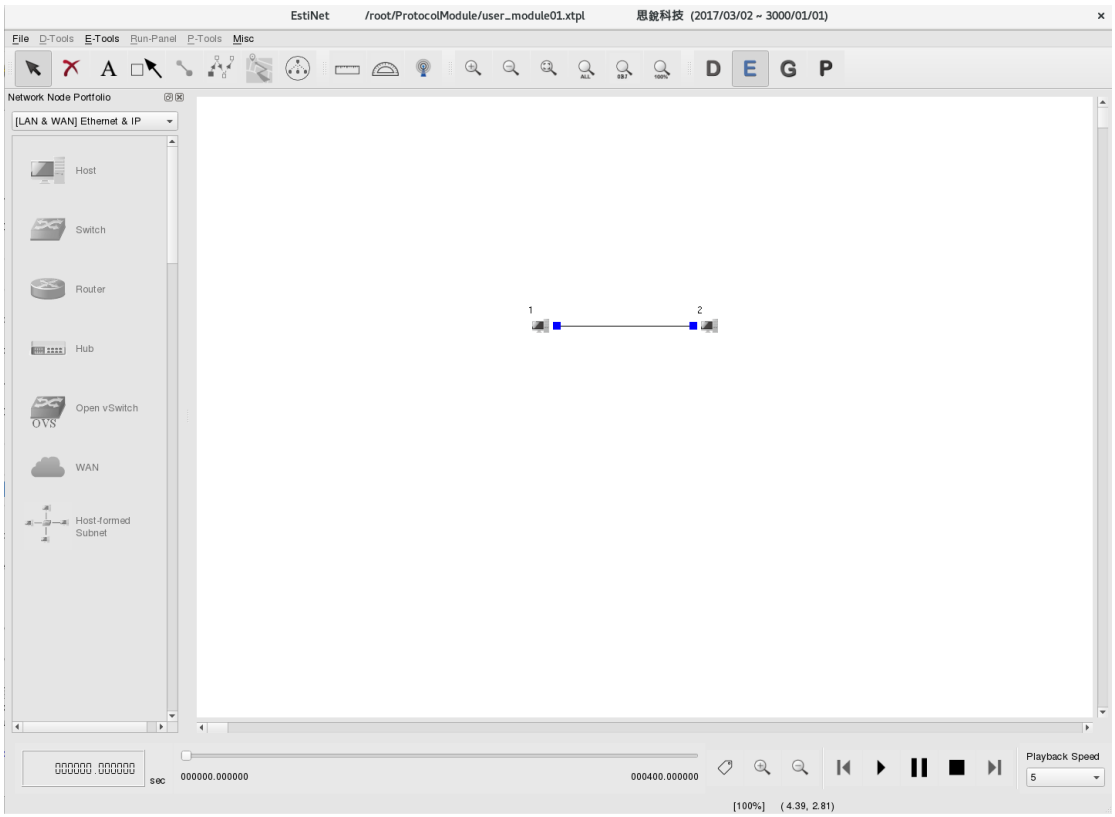
MDF 的全名為 Module Description File，其目的是為了將模組內的參數可以透過 GUI layout 的畫面來設定，使用者在 layout 設定完這些參數後，經由 GUI 程式切換到 G Mode 時，就會寫入 sim/interface_and_medium_setting/general/目錄中的 if_and_medium_conf 設定檔中。

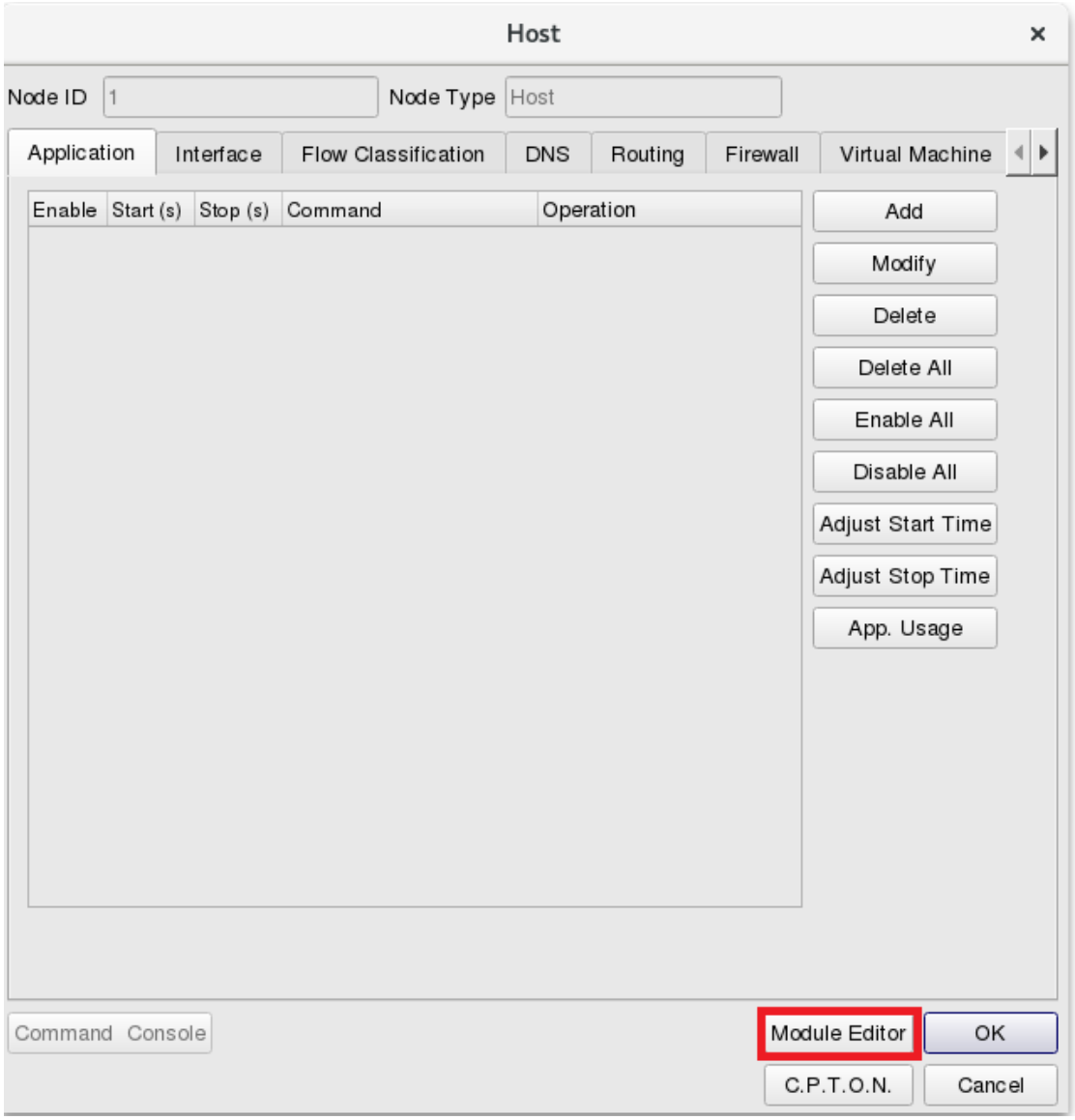
因此 MDF 是對開發模組的人，是一個相當重要的工具。MDF 可以設定自己的 Layout 與參數來便利使用者與開發人員，Layout 常用的物件如 RADIOBOX、TEXTLINE、CHECKBOX、LABLE 等等。

練習 1-1：修改 MDF (搭配 GUI)、修改 Protocol Stack

- 建立範例拓撲

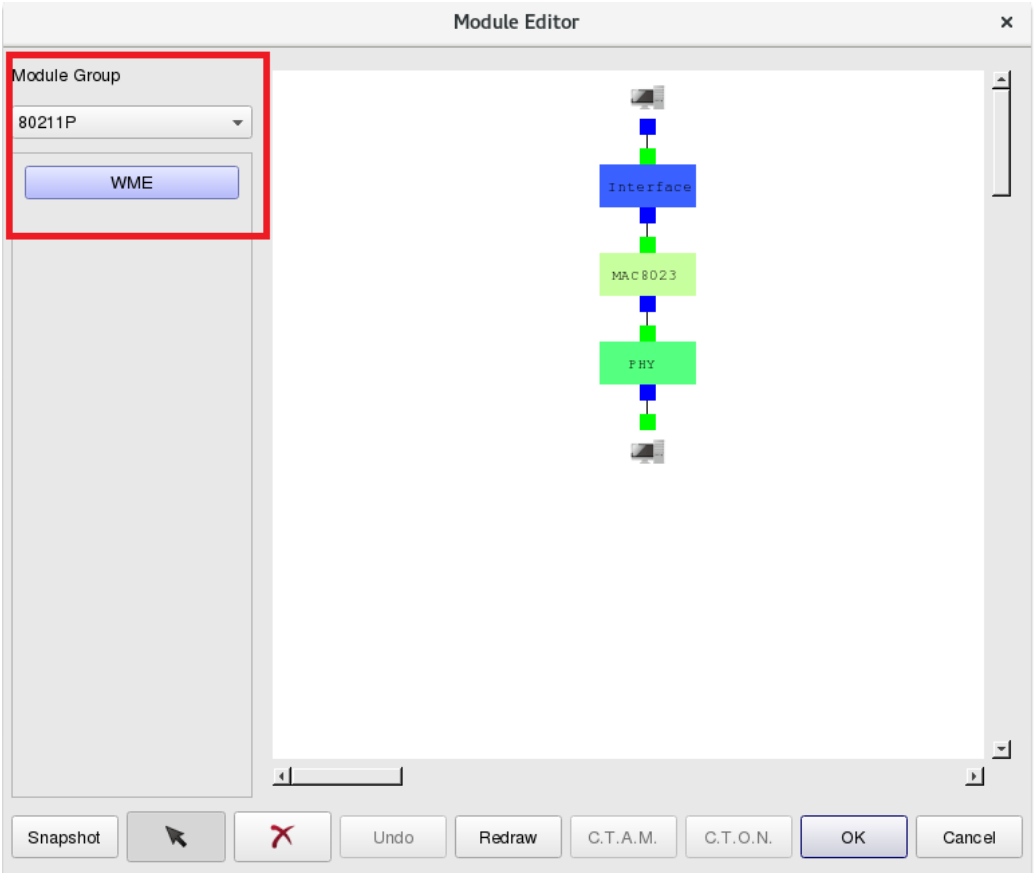
  在 GUI 的 D Mode 建立如下圖的拓撲，是 Host1 連接 Host2 的拓撲，接著轉換到 E Mode 時存檔為 user_moduler01。



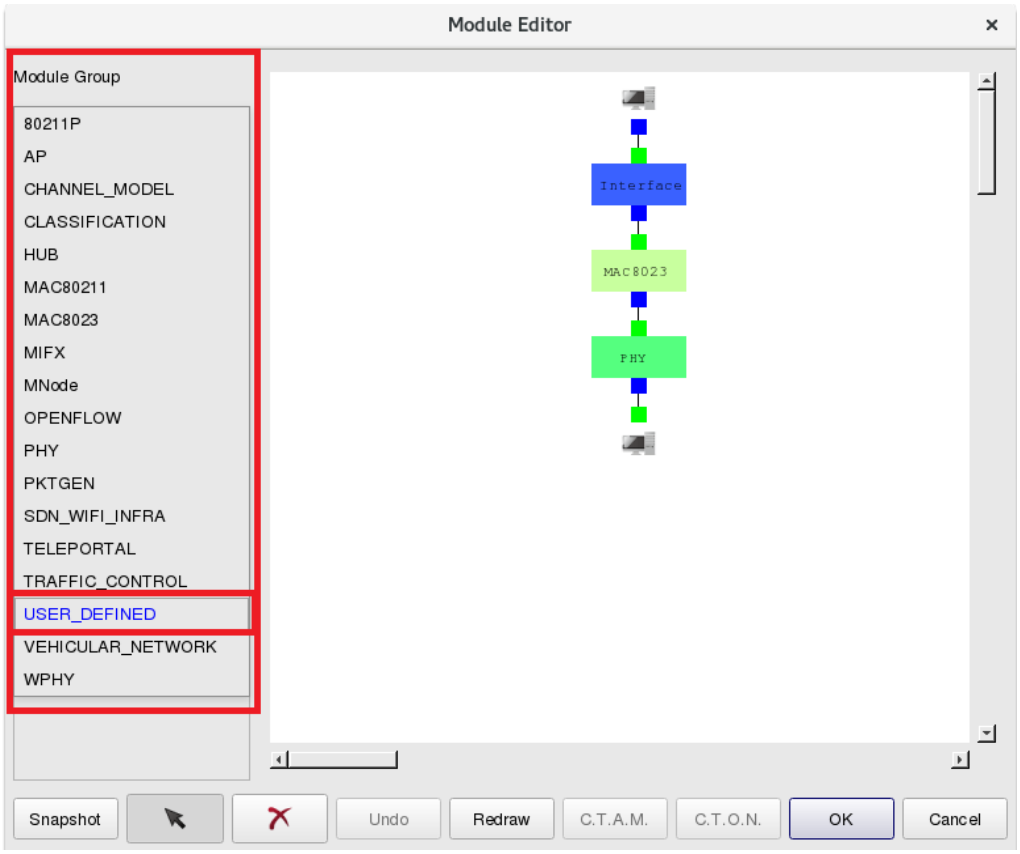


在 E Mode 點選兩下 Host1，再點選"Module Editor"按鈕，可以看到這個節點中的 Protocol Stack。

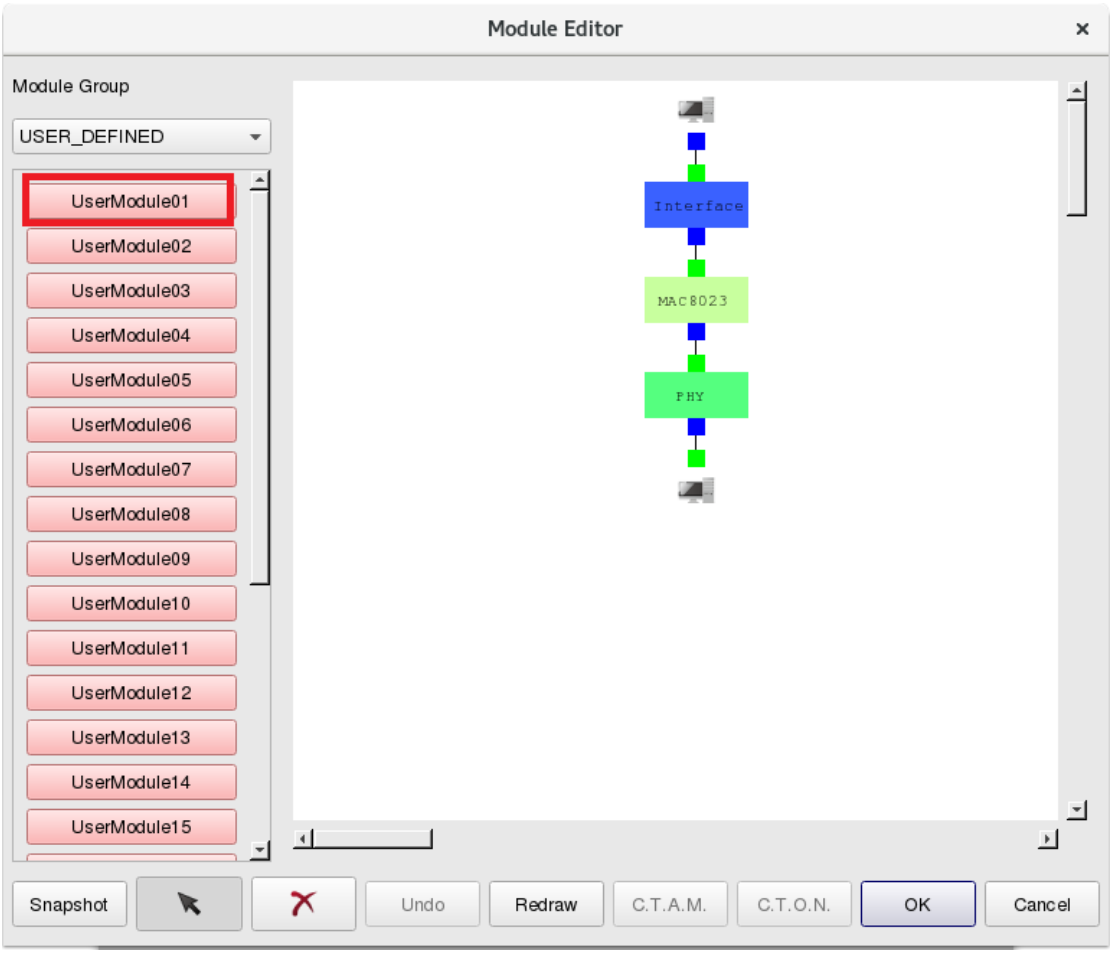
首先加入一個使用者可以自定義的模組，準備在這個模組中加入使用者的程式碼。所以要將這個模組加入 Protocol Stack 中。首先看到左側有一個下拉式選單"Module Group"，GUI 負責管理跟分類這些模組，如第一個放的是 80211P 的選項中，裡面就有 WME 的模組。



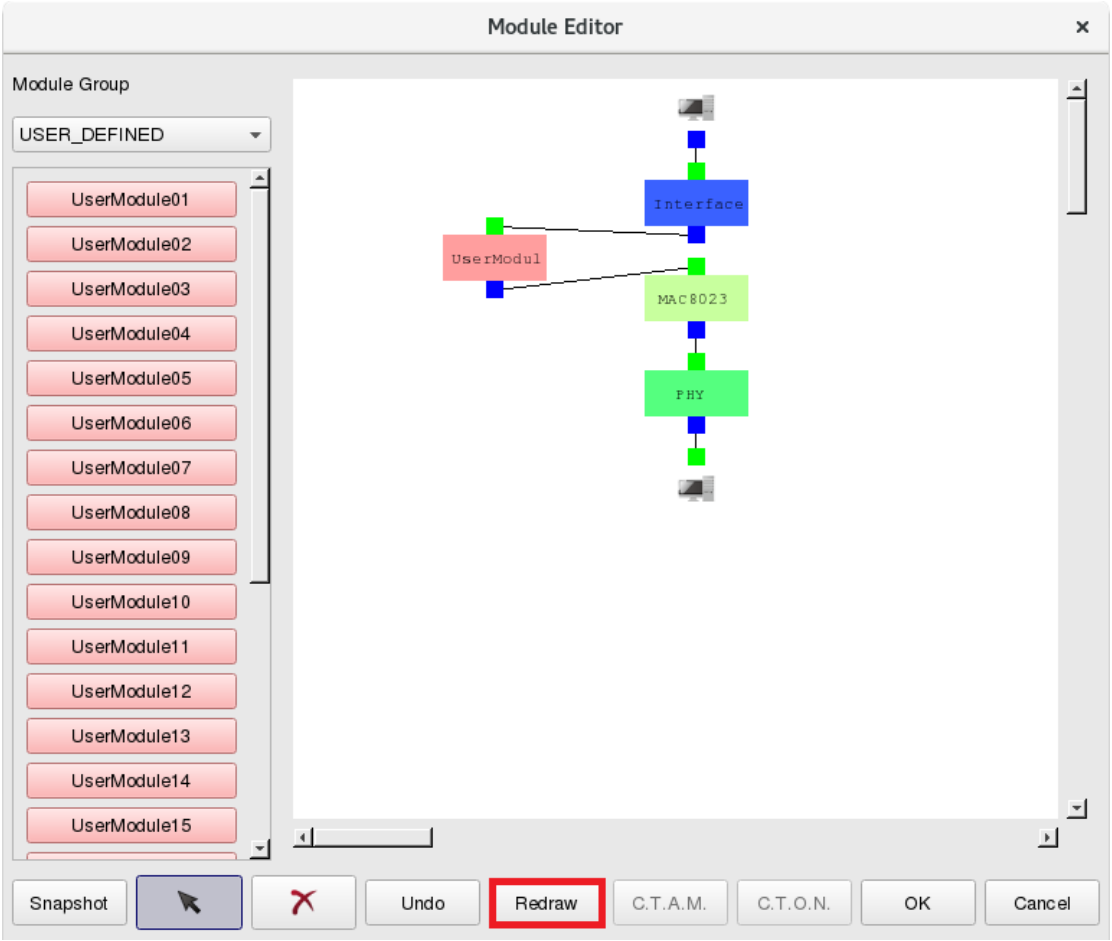
請選擇"USER_DEFINED"模組 GROUP。



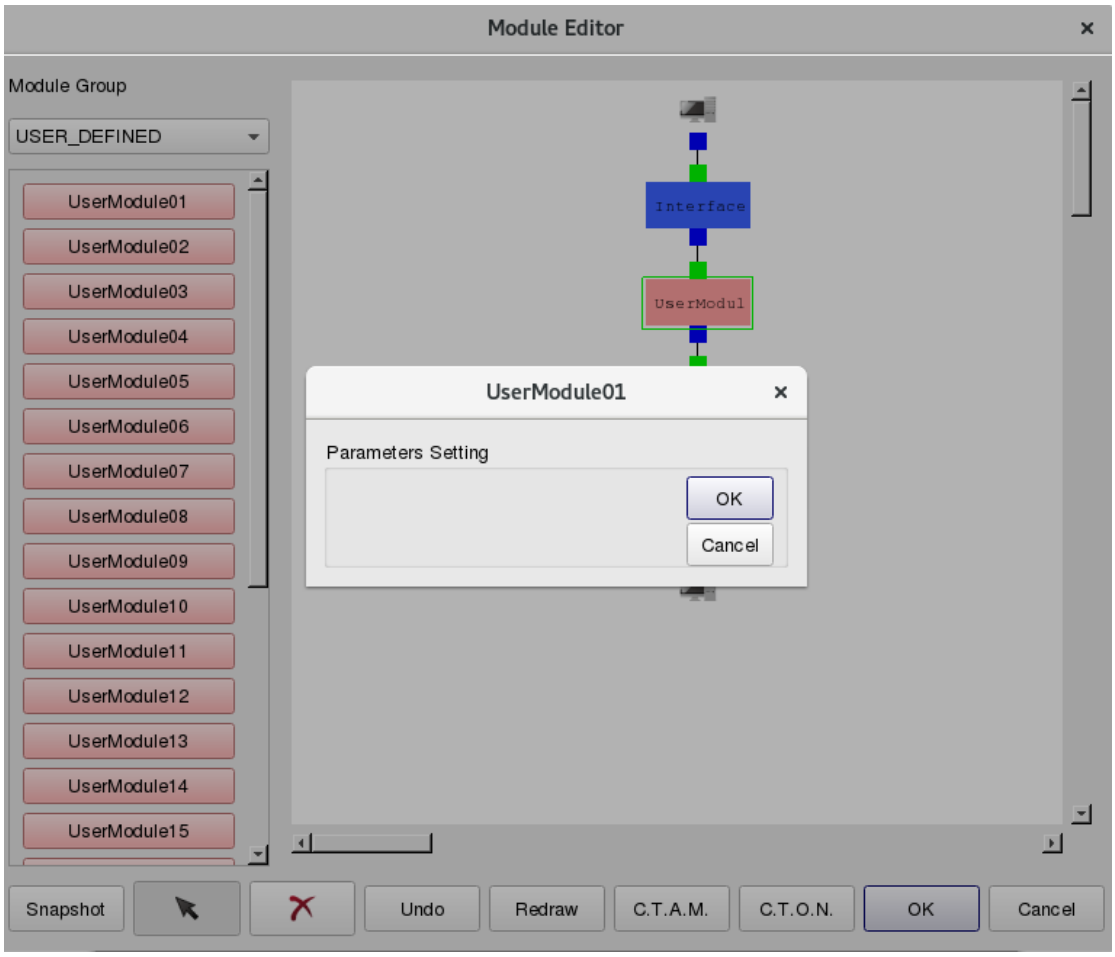
選擇"USER_DEFINED"模組 GROUP 之後，可以看到如下圖共有 25 個 UserModule 可以讓使用者自行定義，這 25 個模組是免費提供給使用者使用。因此點選第一個免費提供的"UserModule01"模組。



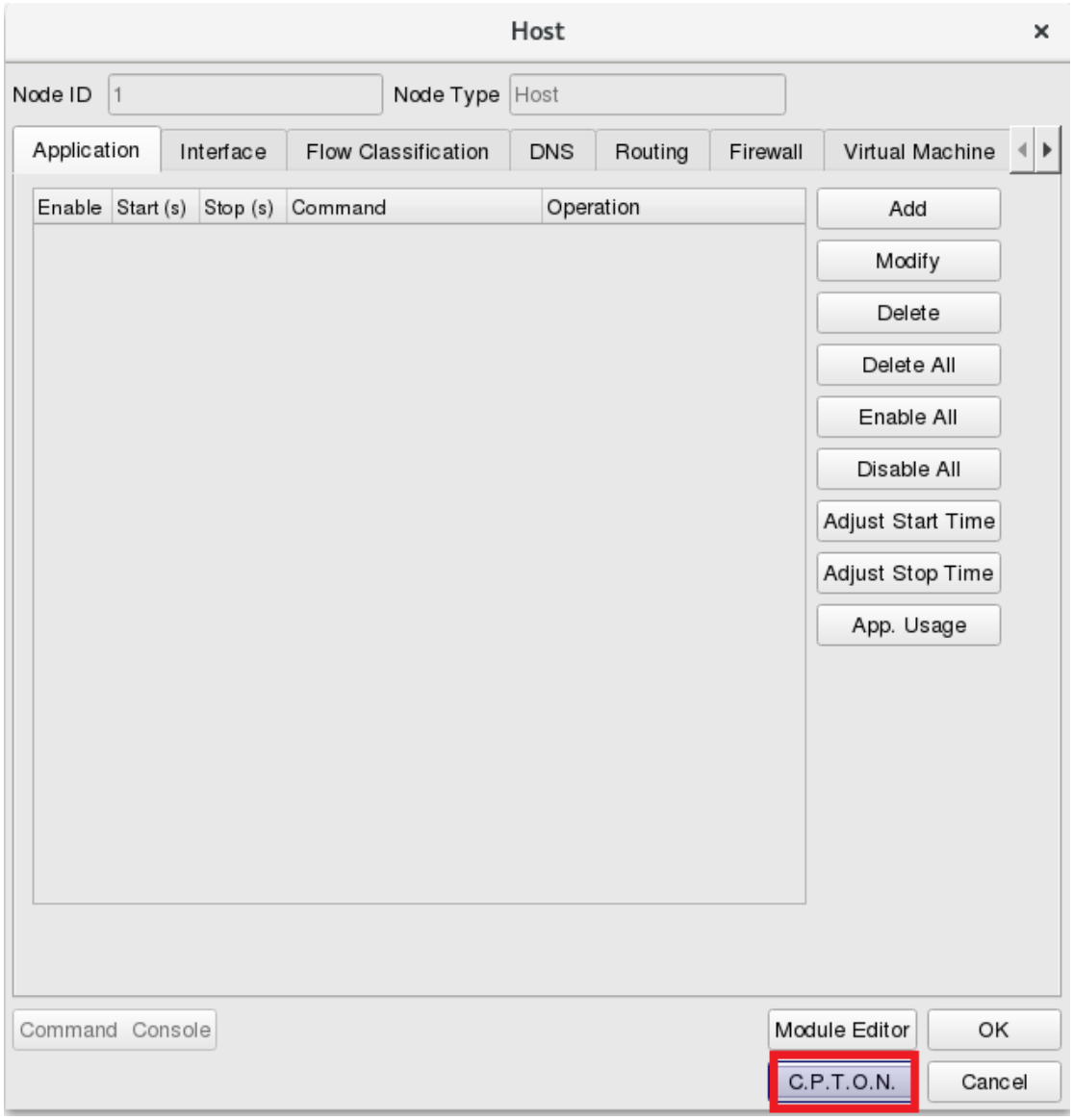
點選後，直接放到白色的畫布上，將"UserModule01"模組放入到 Interface 跟 MAC8023 之間，用下面按鈕"X"，將 Interface 與 MAC8023 之間的連線刪除，再用箭號的按鈕，將 UserModule01 的上面綠色的接頭接到 Interface 下面的藍色接頭；UserModule01 的下面藍色的接頭接到 MAC8023 上面的綠色接頭，就完成將 UserModule01 加入到 HOST1 中的 Protocol Stack 了！(畫完後，可以按下"ReDraw"按鈕重新整理 Protocol Stack)



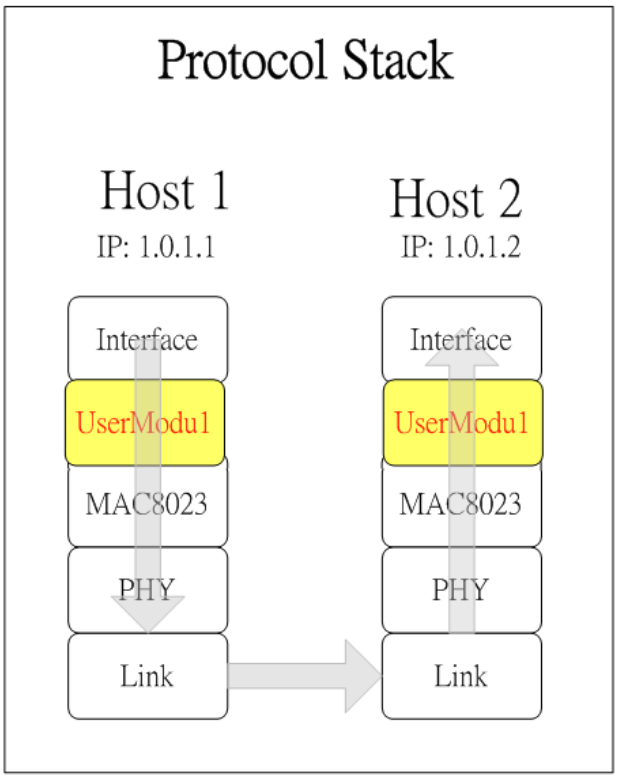
點選兩下這個 UserModule01 模組，會發現這個模組目前是空的，稍後會說明如何修改 MDF 來加入 Layout 物件(如 TEXTLINE、CHECKBOX 等)來方便使用者/開發者填入參數。



同樣地 Host2 也要是一模一樣的 Protocol Stack，因此按下"OK"按鈕後回到 Host1 的設定畫面，接著按下"C.P.T.O.N."按鈕(Copy the Node's Protocol Stack to Other Nodes with the Same Type)，將 Host1 的 Protocol Stack 複製到其它的 Host 上(在此拓撲即為 Host2)。



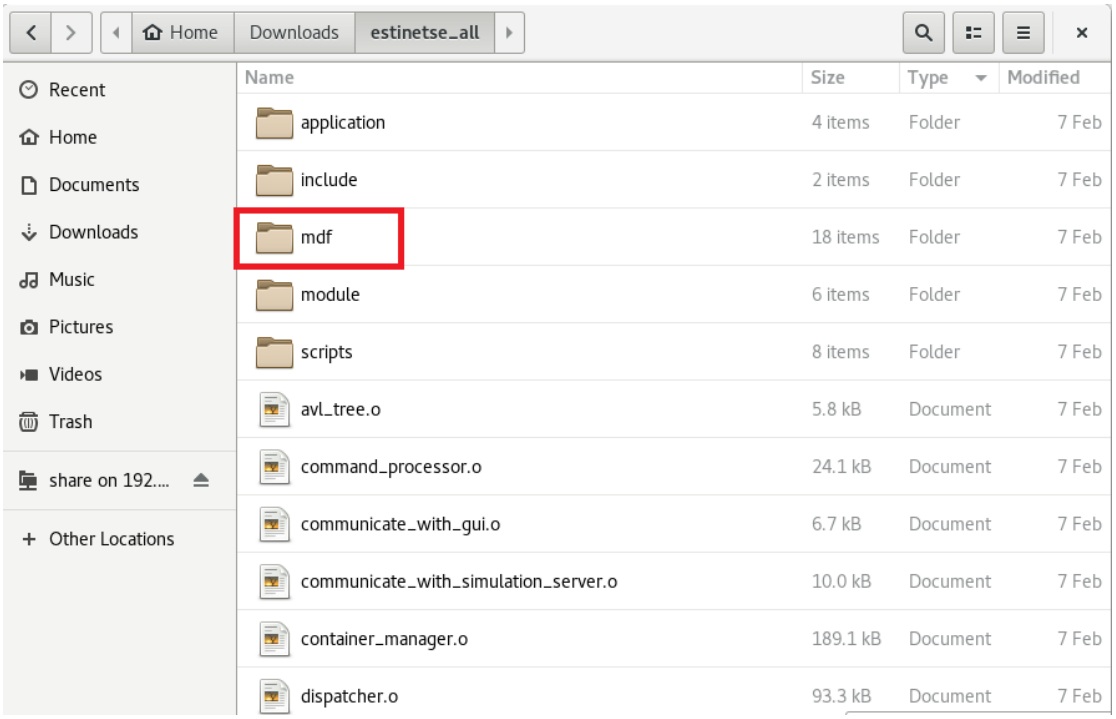
拓撲即設置完成。本手冊後面的拓撲也會由此架構延伸，整個的 Protocol Stack 如下圖所示：



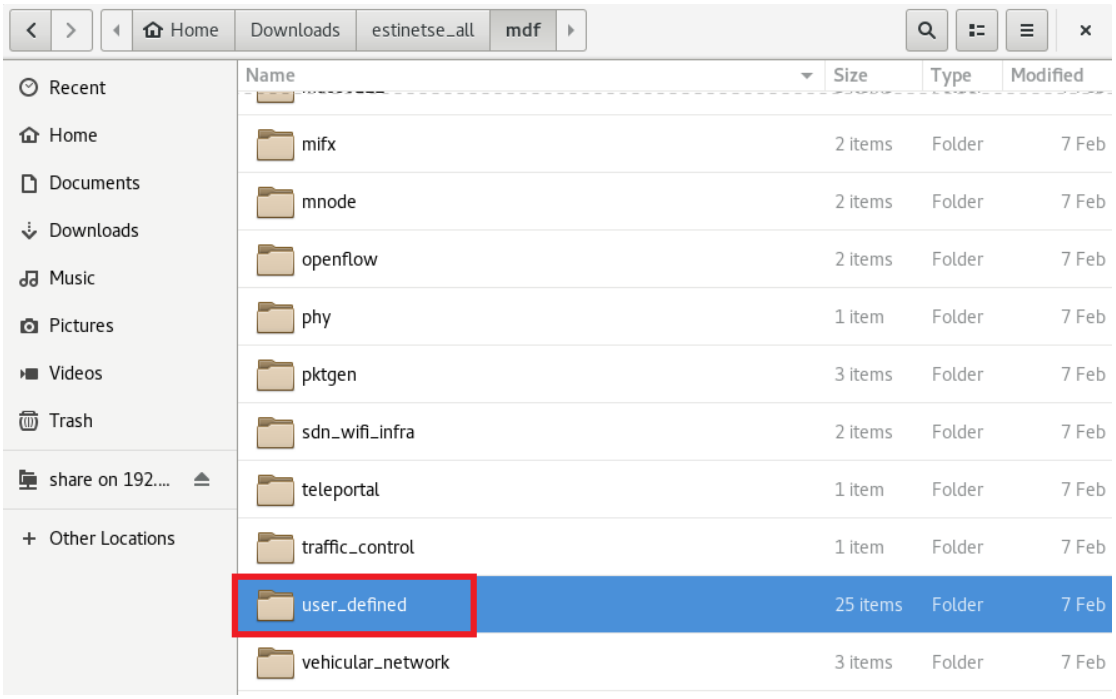
- 介紹 MDF 的結構

  接著要開始來編寫 UserModule01 這個模組的 MDF 檔，希望可以加入 TEXTLINE 來填入兩個參數，一個是"My Number"；另一個是"My String"。

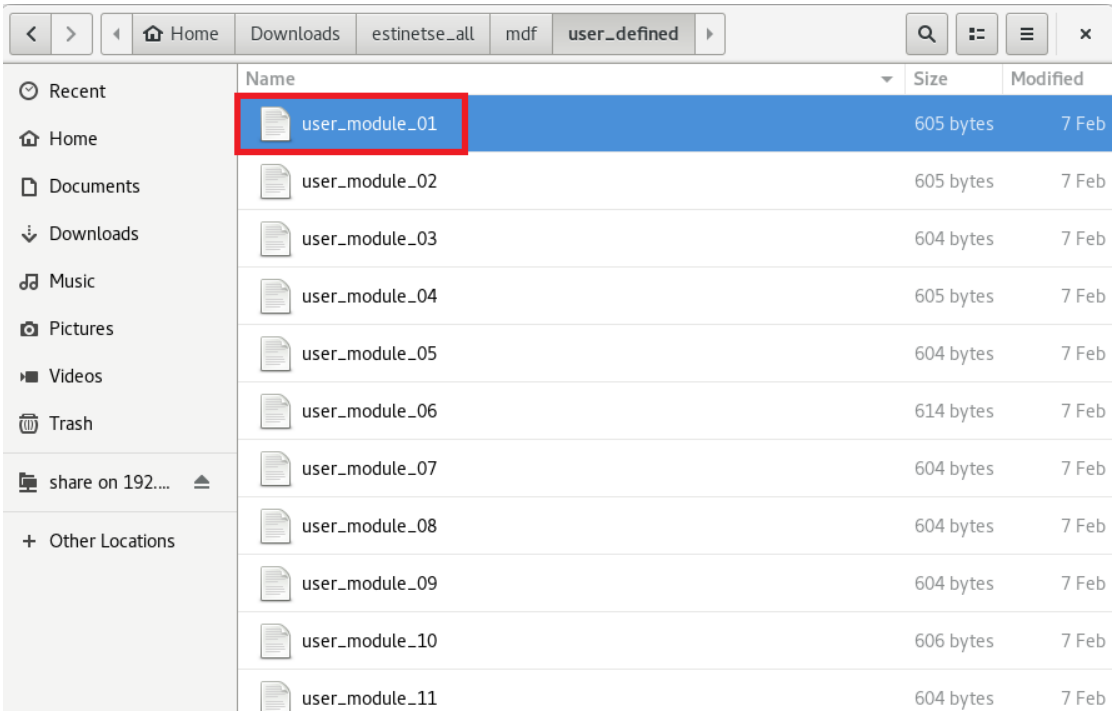
  因此需要修改 MDF 檔案，首先開啟原始檔目錄中(因使用者放置的路徑都不同，請使用者查看自己所放置的原始碼位置)，有一個 mdf 的目錄，如下圖所示。



進入 mdf 目錄後可以看到一個 UserDefined 的目錄，如下圖所示：



UserDefined 目錄裡面共有 25 個 UserModule 的 MDF 檔案，因拓撲中使用的是 UserModule01 的模組，因此開啟此檔案來修改。



開啟 MDF 後如下圖所示，與一般的程式語言的文法並不相同，這是因為 MDF 是專門給 GUI 讀的檔案。GUI 讀取的時機在於開啟 Module editor 按鈕時，就會將所有的 MDF 按照其 Group Nmae(MDF 檔案中的其中一個欄位)進行分類擺在左側；另外當某個模組被點開的時候，GUI 便按照此模組的 MDF 檔案的語法執行命令。

MDF 的基本架構：

MDF 檔的首尾會被"**ModuleSection**”以及“**EndModuleSection**”所包括。中間主要有三大區塊，分別為"**HeaderSection**"、"**InitVariableSection**"、"**ExportSection**"。這三個區塊結尾時的關鍵字分別為"**EndHeaderSection**"、"**EndInitVariableSection**"、"**EndExportSection**"。

HeaderSection 區塊是定義模組的名稱、以及模組所屬的 Group(GUI 分類用)，以及網路的類型、參數的宣告等等。

InitVariableSection 區塊是定義 GUI 的物件的呈現，例如按鈕、TEXTLINE、RADIOBOX、CHECKBOX、GROUP 物件等等…。

ExportSection 區塊則是定義用來當模擬運行時，GUI 可以透過兩種所定義的物件跟模擬引擎透過 IPC(inter-process communication)的方式拿取資料或設定資料。在下一章會介紹這個區塊的使用方式。

user_module_01

```
ModuleSection
    HeaderSection
        ModuleName      UserModule01

        GroupName       USER_DEFINED
        Introduction        "Empty module for development"

    EndHeaderSection

    InitVariableSection
        Caption         "Parameters Setting"
        FrameSize       320 90

        Begin BUTTON        b_ok
            Caption     "OK"
            Scale       250 17 60 30
            ActiveOn    ALL_MODE
            Action      ok
            Comment     "OK Button"
        End

        Begin BUTTON        b_cancel
            Caption     "Cancel"
            Scale       250 49 60 30
            ActiveOn    ALL_MODE
            Action      cancel
            Comment     "Cancel Button"
        End
    EndInitVariableSection

    ExportSection
        Caption         ""
        FrameSize       0 0
    EndExportSection
EndModuleSection
```

範例中希望新增兩個參數，因此首先先在 **HeaderSection** 中宣告兩個參數，一個是"MyNumber"；另一個是"MyString"在 HeaderSection 中，如下圖紅字所示：

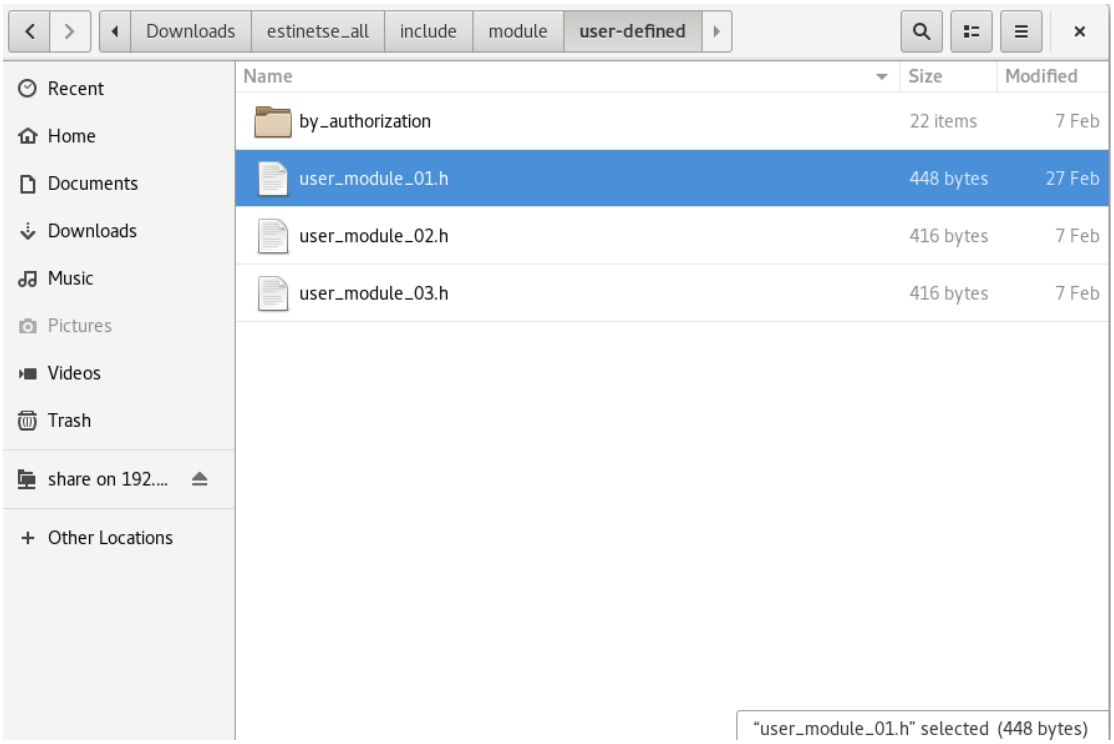
```
ModuleSection
    HeaderSection
        ModuleName          UserModule01
        GroupName           User_Defined
        Introduction        "This is a user-defined module."
        Parameter           myNumber 300 local
        Parameter           myString 111.111.111.111 local
    EndHeaderSection


    InitVariableSection
```

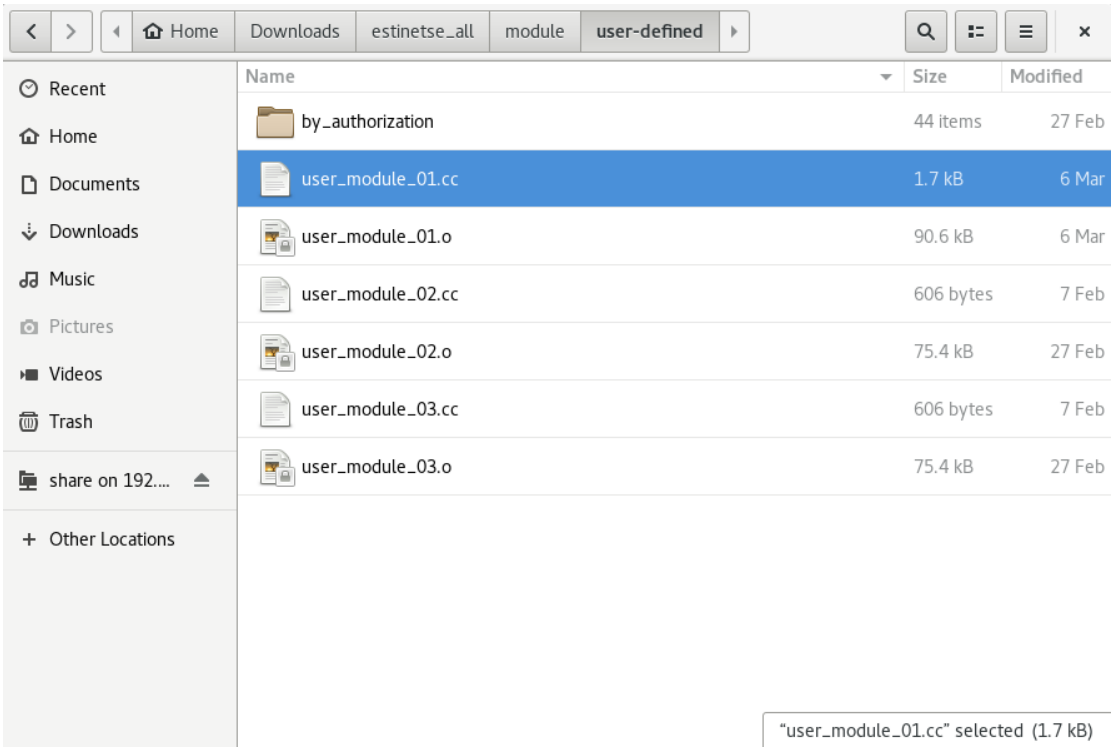
```
            Caption             "Parameters Setting"
            FrameSize           320 90

            Begin BUTTON        b_ok
                Caption         "OK"
                Scale           250 17 60 30
                ActiveOn        ALL_MODE
                Action          ok
                Comment         "OK Button"
            End

            Begin BUTTON        b_cancel
                Caption         "Cancel"
                Scale           250 49 60 30
                ActiveOn        ALL_MODE
                Action          cancel
                Comment         "Cancel Button"
            End
        EndInitVariableSection

        ExportSection
            Caption             ""
            FrameSize       0 0
        EndExportSection
EndModuleSection
```

新增參數一開始的識別字是 Parameter, 接著空白後接參數名稱，接著再輸入預設值，最後一個識別字是 local，最後一個識別字是 GUI 專用，可以參考附錄 A 細節。

完成之後，就完成 MDF 檔中新增參數的設定。

接著到模擬引擎中的 USER MODULE 01 模組來設定讀取 MDF 中的設定值。每個模組中都分別有 head 檔跟 cc 檔，user_module_01.h 檔案位置是在原始碼的目錄下的 include/module/user-defined/user_module_01.h；另外 user_module_01.cc 檔案位置是在原始碼的目錄下的 module/user-defined/user_module_01.cc
如下兩張圖所示：





在 user_module_01.h 檔中宣告兩個變數準備將 MDF 的兩個參數透過 if_and_medium_conf 檔讀入進來，宣告一個整數 Number、一個字串指標 String。如下紅字所示：

```
#ifndef __user_module_01_h__
#define __user_module_01_h__

#include <event.h>
#include <object.h>

class UserModule01 : public NslObject {

 private:
      int               Number;
      char              *String;

 public:

      UserModule01(u_int32_t type, u_int32_t id, struct plist* pl, const char *name);
      ~UserModule01();

      int          init();
      int          command(int argc, const char *argv[]);
      int          recv(ePacket_ *pkt);
      int          send(ePacket_ *pkt);

};

#endif   /* __user_module_01_h__ */
```

而在 user_module_01.cc 檔的建構子中，透過 vBind_int 和 vBind_char_str 這兩個 EstiNet 所提供的 API，將 MDF 的變數值與模組中的變數銜接起來，第一個參數放 MDF 中定義的參數名稱，第二個則是在模組中來承接 MDF 參數的 C++變數。用法如下紅字所示。而在 init 函數中，將這兩個變數印出來(列印加入 ANSI escape codes 控制文字顏色輸出為粗體紅色，背景黑色)

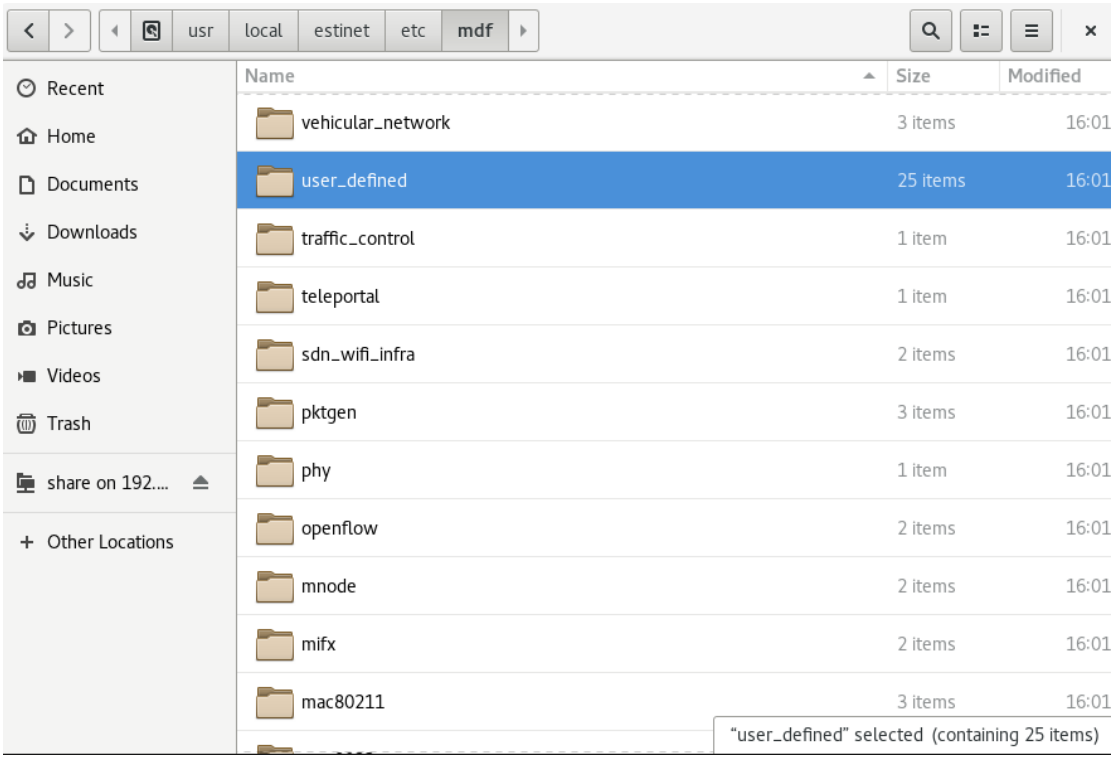
```
#include <stdlib.h>
#include <estinet_api.h>
#include <module/user-defined/user_module_01.h>

MODULE_GENERATOR(UserModule01);
```

```
UserModule01::UserModule01(u_int32_t type, u_int32_t id, struct plist* pl, const char
*name): NslObject(type, id, pl, name) {
        vBind_int("myNumber", &Number);
        vBind_char_str("myString", &String);
}

UserModule01::~UserModule01(){}

int UserModule01::init() {


/* exercise 1 */
  printf("\e[1;31;40m\nExercise 1: myNumber = %d, myString = %s\e[m\n",
     Number, String);

        return(NslObject::init());
}

int UserModule01::command(int argc, const char *argv[]) {
        return(NslObject::command(argc, argv));
}

int UserModule01::recv(ePacket_ *pkt) {
        return(NslObject::recv(pkt));
}

int UserModule01::send(ePacket_ *pkt) {
        return(NslObject::send(pkt));
}
```

接著開啟終端機到原始碼的目錄下，執行 **make** 來編繹所有的原始碼。

再執行 **make install**，會將模擬引擎(estinetse)的 binary 檔以及剛剛設定後的 mdf 檔，搬到 GUI 預設指定讀入的路徑(bin 檔會放到

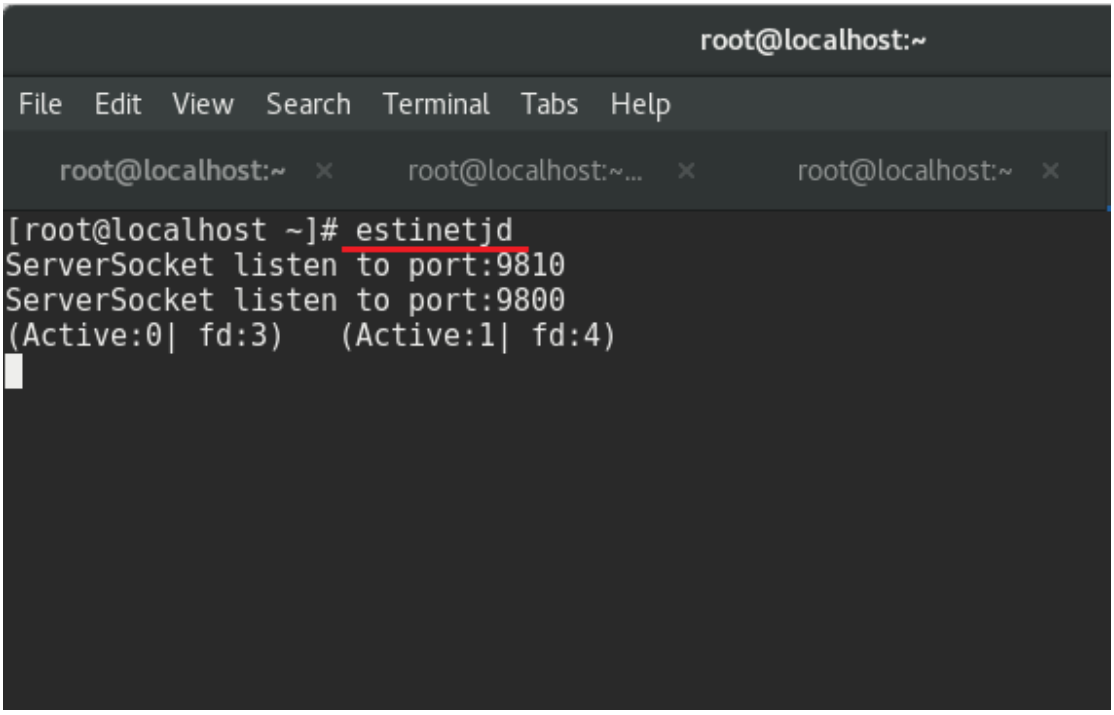
/usr/local/estinet/bin ，mdf 檔則是會放到/usr/local/estinet/etc/mdf)，
如下圖所示：



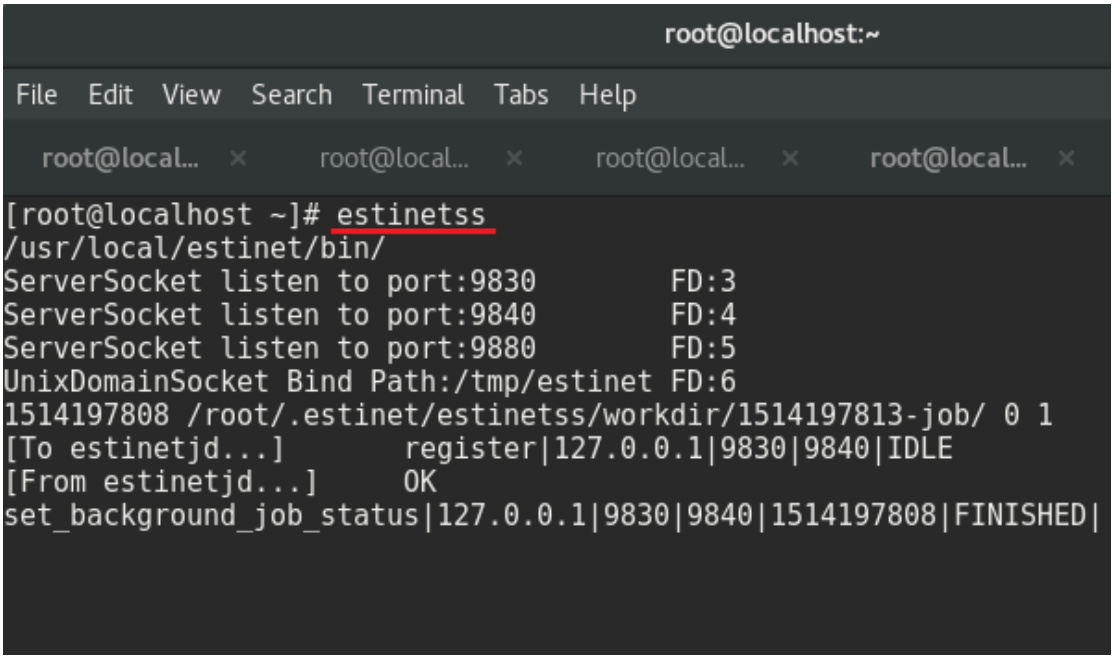
注意：在執行 make install 前，要先關閉 GUI，因為加入 MDF 參數這個動作，是在 GUI 一開始啟動時，就會將所有 MDF 的參數給讀入，如果中途加入參數，則必須關閉 GUI 再重新啟動。如果只是設定 Layout 的寬度、高度或是加入物件等，則可以不用重開 GUI，GUI 只要重新點入模組，就可以看到寬度、高度改變的效果。

完成上述步驟後，編譯完成，並且 install 完成。

接著開啟三個終端機：
第一個終端機執行 estinetjd，如下圖所示：



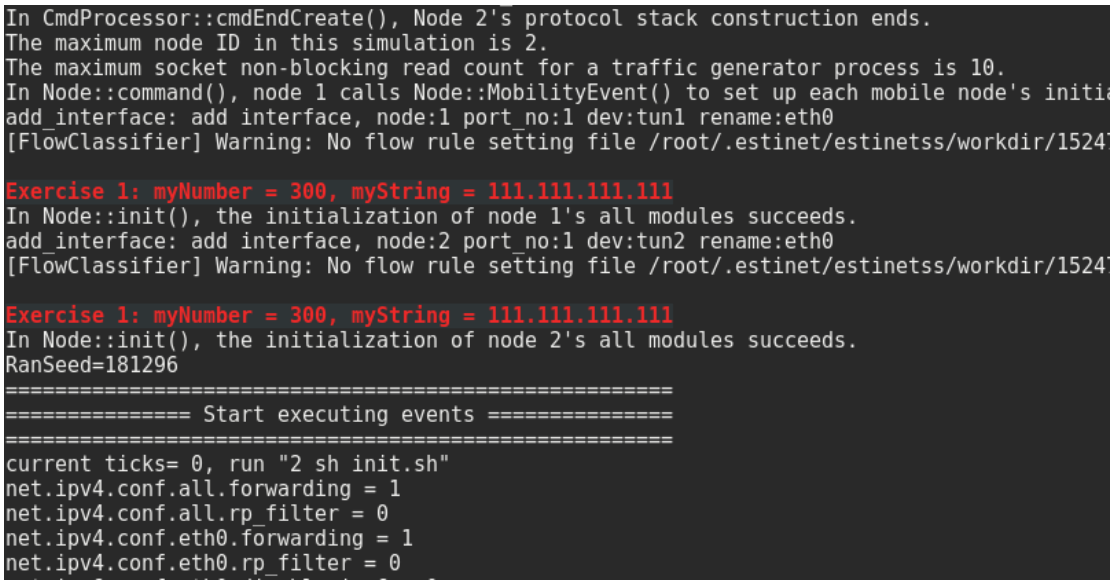
第二個終端機執行指令 **estinetss**



第三個終端機執行 GUI 指令”**estinetgui**”：
首先將剛剛設定 Protocol Stack 的 GUI 存檔後關閉。
重新再開啟 GUI，在終端機輸入 estinetgui，接著在開啟此拓撲 (user_moduler01.xtpl)，並切換到 **G** 模式執行模擬。

注意 estinetss 的終端機視窗中的結果。

因為有兩個節點，且都有放入 User module01 的模組，該模組的 init 函數會列印參數值，因此畫面上有兩次列印結果，如下圖所示。



- 修改 Layout

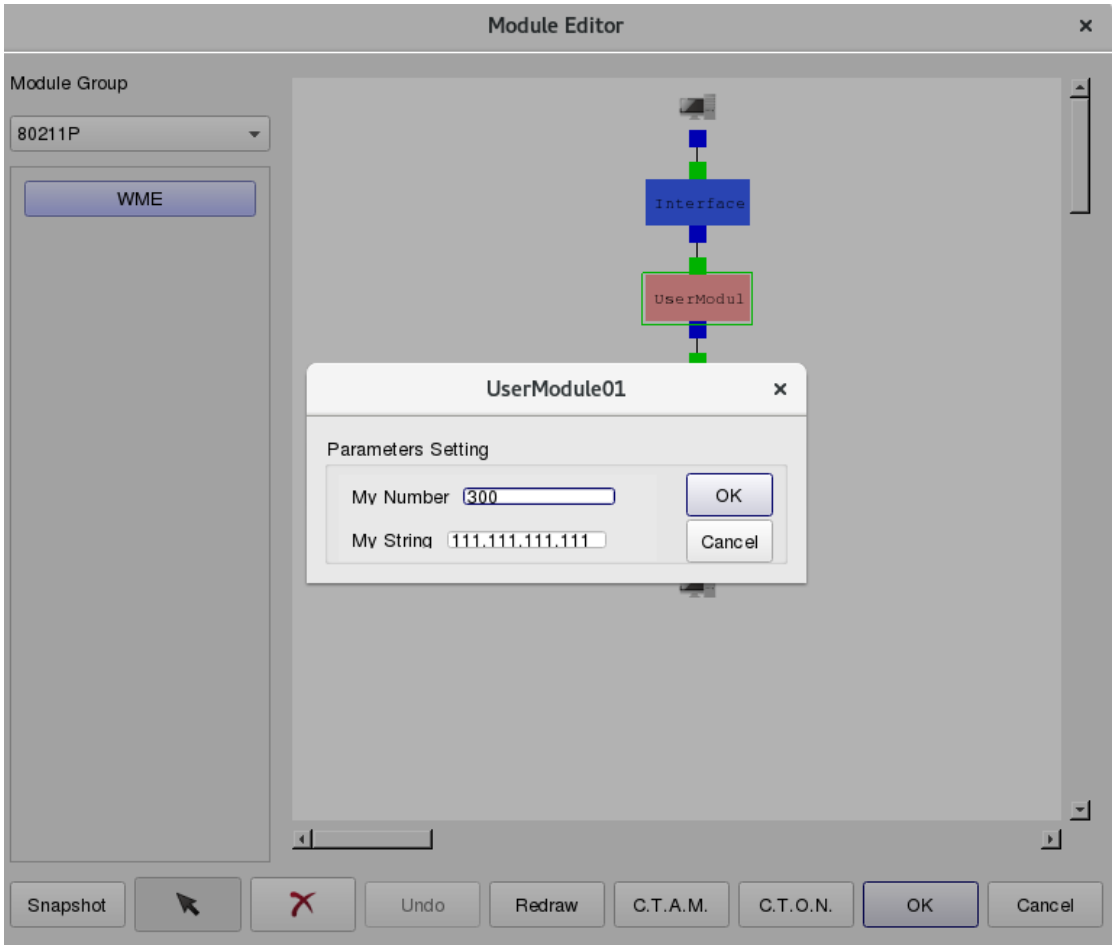
  上面的 estinetss 顯示的結果是 MDF 參數的預設值。如果要透過 GUI 物件輸入數值的話，就需要在 MDF 中使用 TEXTLINE 跟 LABEL 等物件。

```
ModuleSection
    HeaderSection
        ModuleName          UserModule01
        GroupName           User_Defined
        Introduction        "This is a user-defined module."
        Parameter           myNumber 300 local
        Parameter           myString 111.111.111.111 local
    EndHeaderSection

    InitVariableSection
        Caption             "Parameters Setting"
        FrameSize           320 90

        Begin TEXTLINE      myNumber
            Caption         "My Number "
            Scale           10 18 220 30
            ActiveOn        MODE_EDIT
```

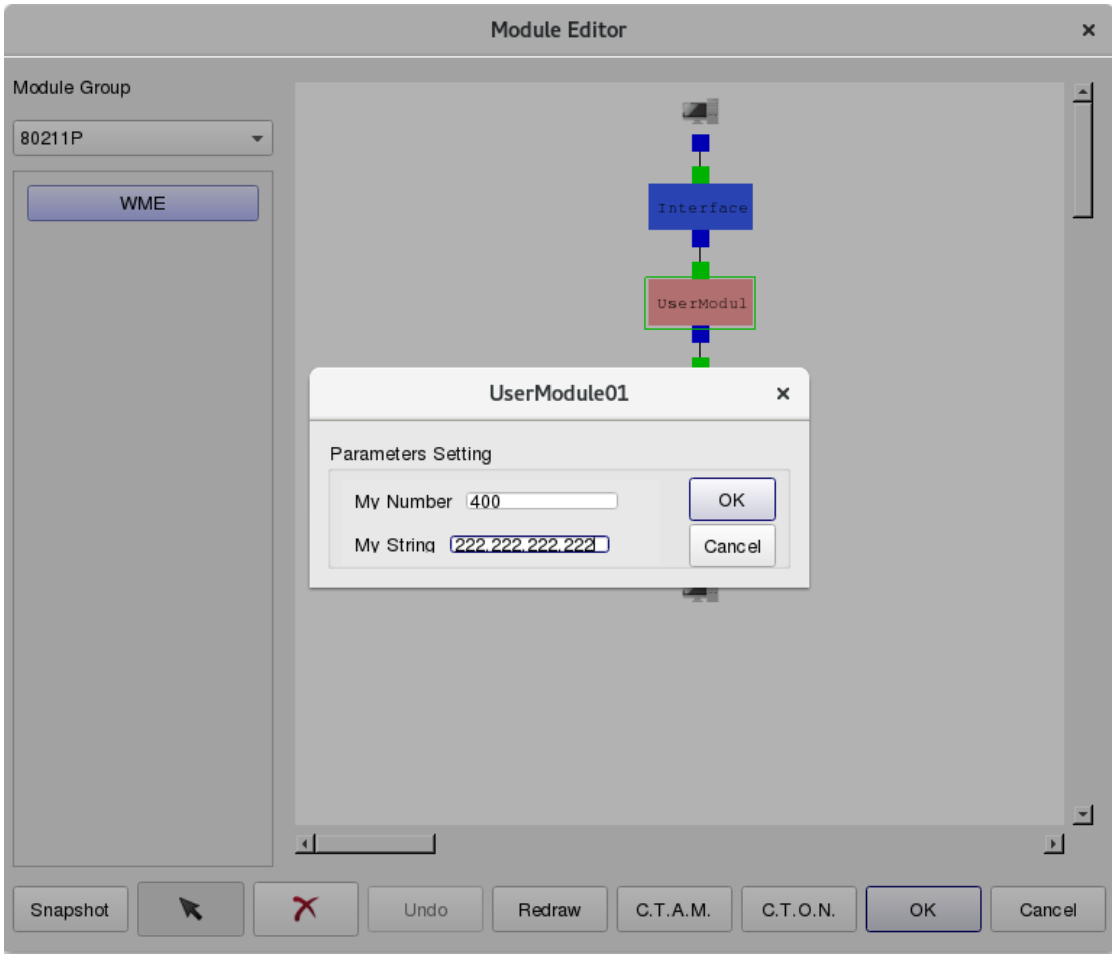
```
                Enabled         TRUE
                Type            INT
                Comment         "An Integer"
            End

            Begin TEXTLINE      myString
                Caption         "My String "
                Scale           10 48 220 30
                ActiveOn        MODE_EDIT
                Enabled         TRUE
                Type            IP
                Comment         "An IP string"
            End

            Begin BUTTON        b_ok
                Caption         "OK"
                Scale           250 17 60 30
                ActiveOn        ALL_MODE
                Action          ok
                Comment         "OK Button"
            End

            Begin BUTTON        b_cancel
                Caption         "Cancel"
                Scale           250 49 60 30
                ActiveOn        ALL_MODE
                Action          cancel
                Comment         "Cancel Button"
            End
        EndInitVariableSection

        ExportSection
            Caption         ""
            FrameSize       0 0
        EndExportSection
EndModuleSection
```

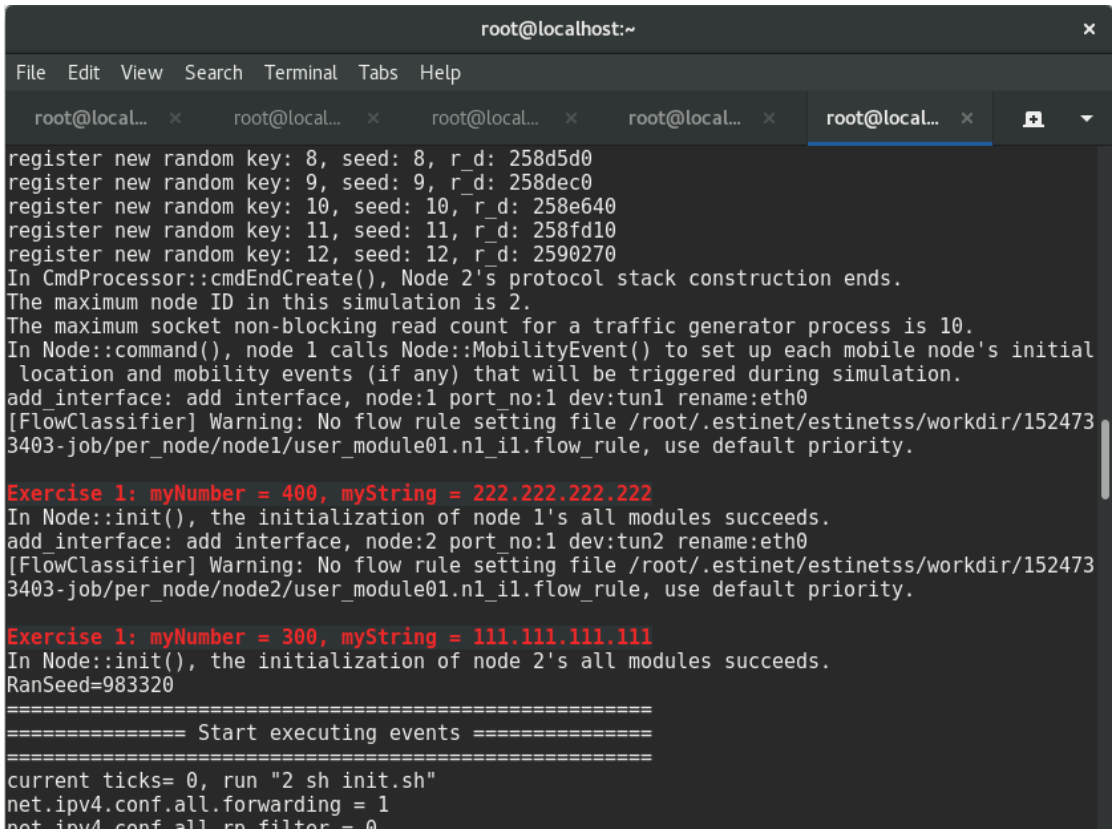
加入 MDF 的兩個 TEXTLINE 物件，在 estinetse_all 目錄下需要執行 make install，將新改好的 MDF 檔搬移到 GUI 讀取的位置。但因沒有新加參數，GUI 不用重新開啟，可以直接到 Module Editor 看 USER MODELE 01 模組加入物件的樣子。

如下圖所示：



在 Host1 上的 My Number 輸入 400，My string 輸入 222.222.222.222 再執行模擬一次，注意 estinetss 視窗，可以看到 Host1 上列印的值改變了，如下圖所示：


Module Editor
Module Group
80211P
WME
Interface
UserModul
UserModule01
Parameters Setting
My Number 400
My String 222.222.222.222
OK
Cancel
Snapshot
Undo
Redraw
C.T.A.M.
C.T.O.N.
OK
Cancel


root@localhost:~
File Edit View Search Terminal Tabs Help
root@local...
root@local...
root@local...
root@local...
root@local...
register new random key: 8, seed: 8, r_d: 258d5d0
register new random key: 9, seed: 9, r_d: 258dec0
register new random key: 10, seed: 10, r_d: 258e640
register new random key: 11, seed: 11, r_d: 258fd10
register new random key: 12, seed: 12, r_d: 2590270
In CmdProcessor::cmdEndCreate(), Node 2's protocol stack construction ends.
The maximum node ID in this simulation is 2.
The maximum socket non-blocking read count for a traffic generator process is 10.
In Node::command(), node 1 calls Node::MobilityEvent() to set up each mobile node's initial location and mobility events (if any) that will be triggered during simulation.
add_interface: add interface, node:1 port_no:1 dev:tun1 rename:eth0
[FlowClassifier] Warning: No flow rule setting file /root/.estinet/estinetss/workdir/1524733403-job/per_node/node1/user_module01.n1_i1.flow_rule, use default priority.
Exercise 1: myNumber = 400, myString = 222.222.222.222
In Node::init(), the initialization of node 1's all modules succeeds.
add_interface: add interface, node:2 port_no:1 dev:tun2 rename:eth0
[FlowClassifier] Warning: No flow rule setting file /root/.estinet/estinetss/workdir/1524733403-job/per_node/node2/user_module01.n1_i1.flow_rule, use default priority.
Exercise 1: myNumber = 300, myString = 111.111.111.111
In Node::init(), the initialization of node 2's all modules succeeds.
RanSeed=983320
==========
Start executing events
==========
current ticks= 0, run "2 sh init.sh"
net.ipv4.conf.all.forwarding = 1

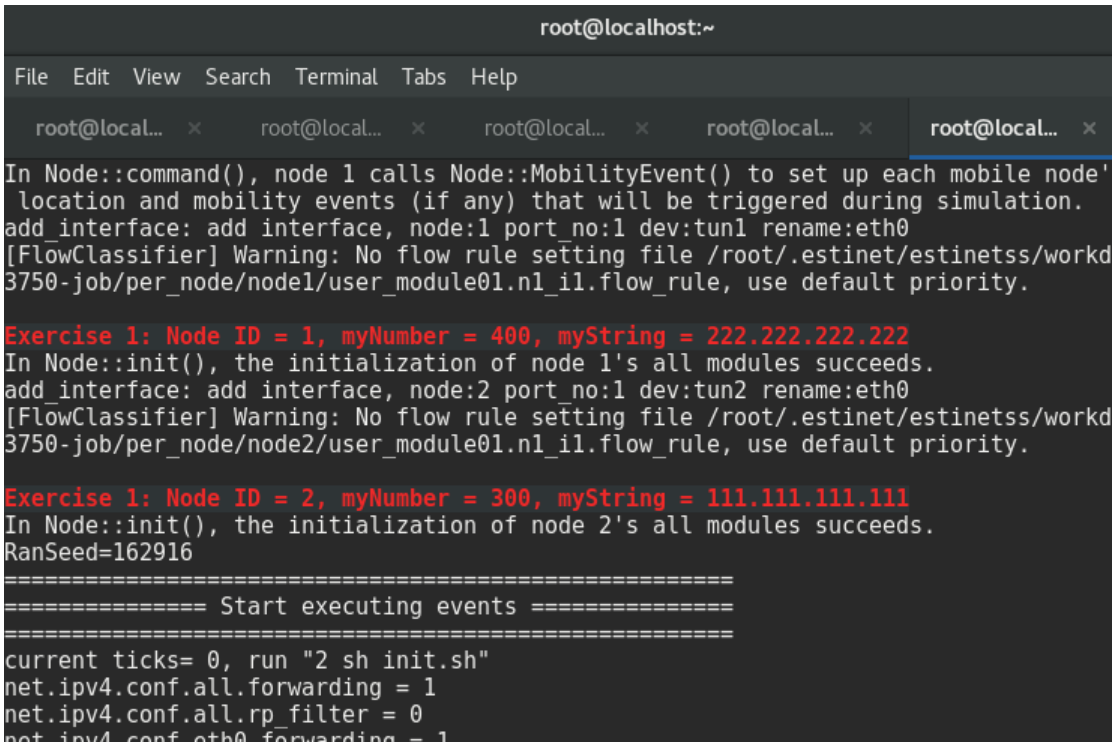
因此可以開發者可以經由 MDF 搭配 GUI，來設定要給模擬引擎的參數。

- 使用 get_nid()

  從列印結果中，其實看不出是哪一個節點印的，可以使用 get_nid()這個 API 來搭配列印：

  ```
  printf("\e[1;31;40m\nExercise 1: Node ID = %d, myNumber = %d, myString = %s\e[m\n", get_nid(), Number, String);
  ```

  列印結果如下圖所示：



- TEXTLINE 的細節

  TEXTLINE 其實還有許多細部的參數可以設定，請查看附錄 B，有 MDF 所有參數完整的介紹。下面簡單介紹常用的部分。

  Caption：

  可以設定此物件的標題

Scale

如上述中，"Scale 10 48 220 30"，四個數值分別代表 X，Y，寬度、高度。左上角的 X,Y 為 0, 0。

ActiveOn

有三個值可以設定，MODE_EDIT, MODE_SIMULATION,ALL_MODE。主要是設定這個物件在哪一個 GUI 的模式下可 active 的。如果設定為 MODE_EDIT，則這個物件在 Edit 模式下是 enable 的。而 MODE_SIMULATION 則是在 G 模式下是 enable 的；ALL_MODE 值則是不論在哪一個模式下都是 enable 的。

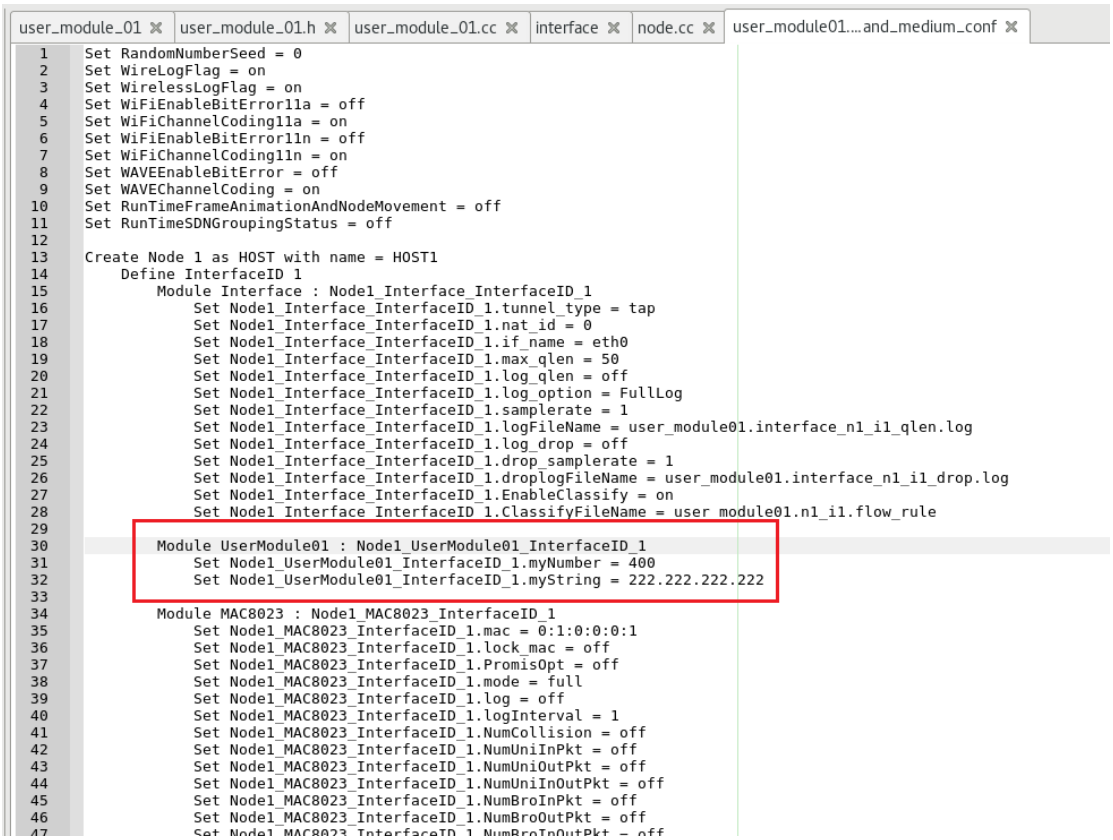
Enabled

這個參數如果是 OFF，則此物件則沒有 enable(灰階)，使用者無法使用。如果為 on，或是在使用在其它模式被 active，才會被 enable，讓使用者使用。

■ 補充：vBind API

上述練習中，模擬引擎中有使用到 vBind 函數，主要是因為要把 GUI 產出的 if_and_medium_conf 中的模組變數，輸入到模擬引擎中的模組中。

上述練習 1-1 中在設定好 MDF 後，在 GUI 執行模擬時，會將 if_and_medium_conf 檔送給模擬引擎，而 if_and_medium_conf 中 UserModule01 的參數值如下所示：



因此，模組要透過 vBind 函式的方式，讀入 if_and_medium_conf 這些值，用 vBind，預期他是什麼資料型態輸入：目前模擬引擎這部分，支援了十種左右的 vBind 資料型別(int、uint8、uint16、bool、float、double、char_str、ip、ipv6、mac)，前者是 MDF 中所變數名，後者是模組中的變數名。下面舉例 vBind_int 這個 API。

vBind_int("myNumber", &Number);

■ 練習 1-2

◆ RADIOBOX & GROUP

接下來使用 RADIOBOX 這個物件練習 Layout。

要特別注意的是使用 RADIOBOX，外層還需要搭配 GROUP 物件，因 RADIOBOX 的長寬高度需要跟 GROUP 物件配合。

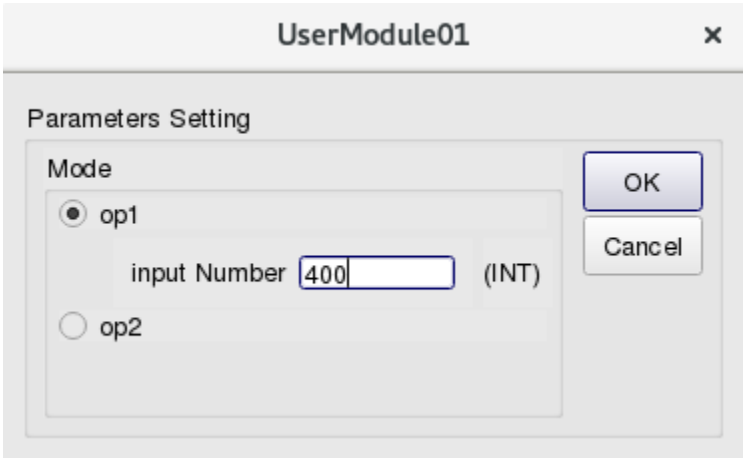
MDF 設計如下所示：

```
ModuleSection
    HeaderSection
        ModuleName          UserModule01
```

```
        GroupName           User_Defined
        Introduction        "This is a user-defined module."
        Parameter           myNumber 300 local
        Parameter           myString string1 local
EndHeaderSection

InitVariableSection
        Caption             "Parameters Setting"
        FrameSize           350 170

        Begin Group         g_radio
            Caption         "Mode"
            Scale           10 15 260 135
            ActiveOn        MODE_EDIT
            Enabled         TRUE

            Begin RADIOBOX      myString
                Option          "op1"
                Enable          myNumber
                Enable          lable1
                OptValue        "string1"
                VSpace          5
                EndOption

                Option          "op2"
                Disable         myNumber
                Disable         lable1
                OptValue        "string2"
                VSpace          40
                EndOption

                Type            STRING
                Comment         "radiobox test"
            End

           Begin TEXTLINE       myNumber
                Caption         "input Number"
                Scale           35 35 180 35
```

```
                    ActiveOn        MODE_EDIT
                    Enabled         FALSE
                    Type            INT
                    Comment         "for test"
                End

                Begin LABEL         lable1
                    Caption         "(INT)"
                    Scale           220 35 35 35
                    ActiveOn        MODE_EDIT
                    Enabled         FALSE
                End
            End

            Begin BUTTON            b_ok
                Caption             "OK"
                Scale               280 17 60 30
                ActiveOn            ALL_MODE
                Action              ok
                Comment             "OK Button"
            End

            Begin BUTTON            b_cancel
                Caption             "Cancel"
                Scale               280 49 60 30
                ActiveOn            ALL_MODE
                Action              cancel
                Comment             "Cancel Button"
            End
        EndInitVariableSection

        ExportSection
            Caption             ""
            FrameSize           0 0
        EndExportSection
EndModuleSection
```

接著將 MDF 檔 install，開啟 GUI 後，執行畫面如下所示：



RADIOBOX 及 GROUP 相關參數的細節，請參考附錄 B。

■ 練習 1-3

◆ CHECKBOX

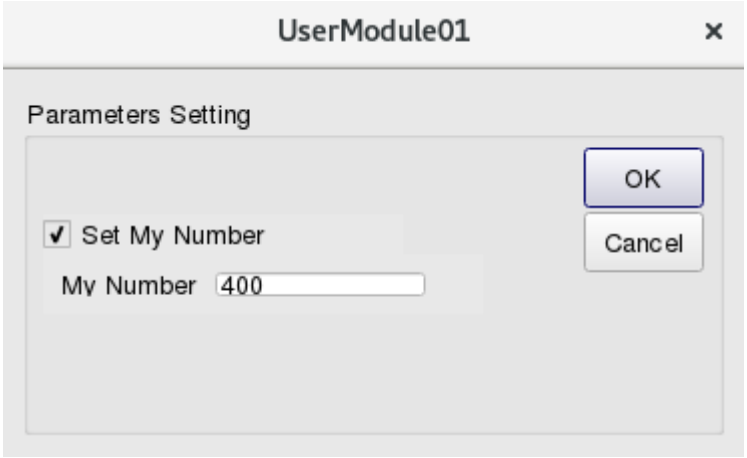
接下來使用 CHECKBOX，請使用者使用下面範例試看看。

```
ModuleSection
    HeaderSection
        ModuleName          UserModule01
        GroupName           User_Defined
        Introduction        "This is a user-defined module."
        Parameter           myNumber 300 local
        Parameter           myString 111.111.111.111 local
    EndHeaderSection

    InitVariableSection
        Caption             "Parameters Setting"
        FrameSize           350 170

        Begin CHECKBOX      check1
            Caption         "Set My Number"
            Scale           10 50 180 20
            ActiveOn        MODE_EDIT
            Enabled         TRUE

            Option          "TRUE"
            OptValue        "on"
```

```
            Enable          myNumber
            EndOption

            Option          "FALSE"
            OptValue        "off"
            Disable         myNumber
            EndOption
            Comment         ""
        End

        Begin TEXTLINE      myNumber
            Caption         "My Number "
            Scale           10 70 220 30
            ActiveOn        MODE_EDIT
            Enabled         FALSE

            Type            INT
            Comment         "An Integer"
        End

        Begin BUTTON        b_ok
            Caption         "OK"
            Scale           280 17 60 30
            ActiveOn        ALL_MODE
            Action          ok
            Comment         "OK Button"
        End

        Begin BUTTON        b_cancel
            Caption         "Cancel"
            Scale           280 49 60 30
            ActiveOn        ALL_MODE
            Action          cancel
            Comment         "Cancel Button"
        End
    EndInitVariableSection

    ExportSection
```

```
        Caption             ""
        FrameSize           0 0
    EndExportSection
EndModuleSection
```

接著將 MDF 檔 install，開啟 GUI 後，執行畫面如下所示：



CHECKBOX 相關參數的細節，請參考附錄 B。

第二章：MDF 與 Run Time Query

本章重點：
(1)、如何使用 RUN TIME QUERY
(2)、第一種 EXPORT SECTION 的物件 -- **ACCESSBUTTON**
(3)、第二種 EXPORT SECTION 的物件 -- **INTERACTIONVIEW**
(4)、介紹 EXPORT 的 API，以及 COMMAND 這個 FUNCTION

下載練習：

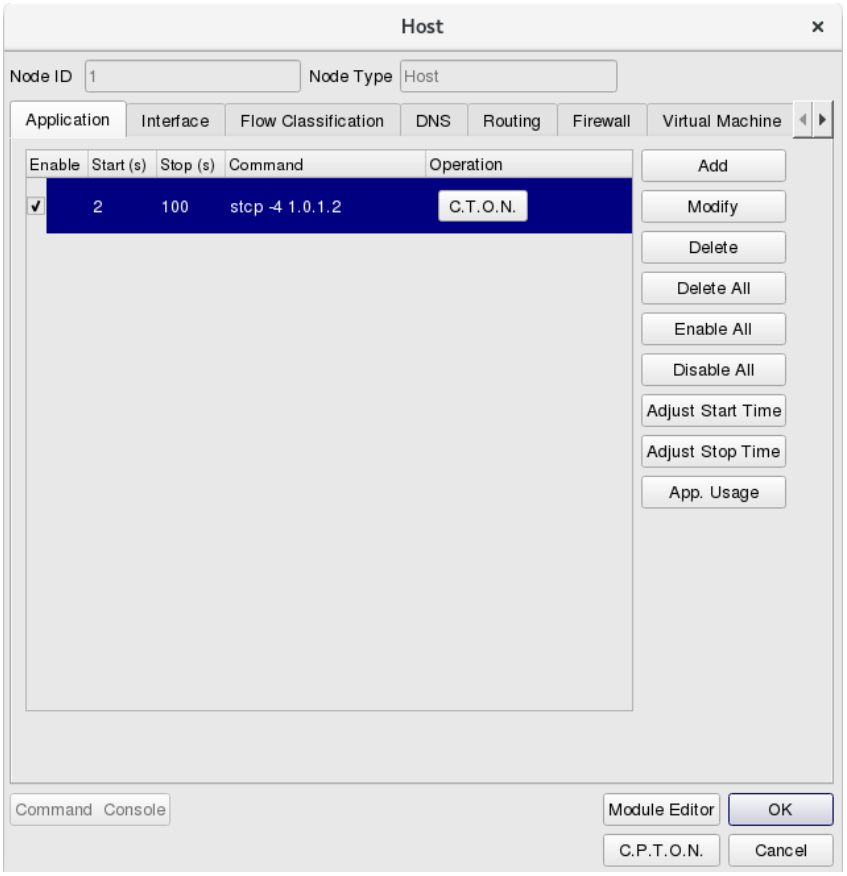

Chapter2.tar.bz2

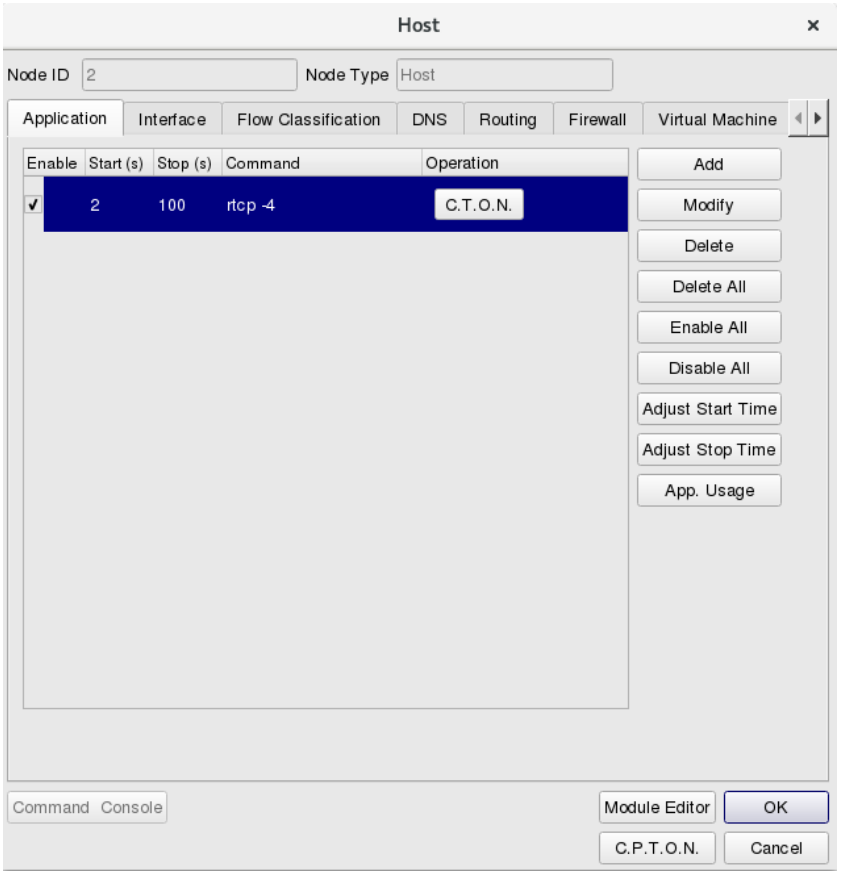
MDF 中可以用來模擬中查詢變數的物件有兩種，分別是 **ACCESSBUTTON**，另一種是 **INTERACTIONVIEW**。這兩種都要搭配模組中的 **COMMAND** 函數以及 **EXPORT** 函數。

- 拓撲設定：

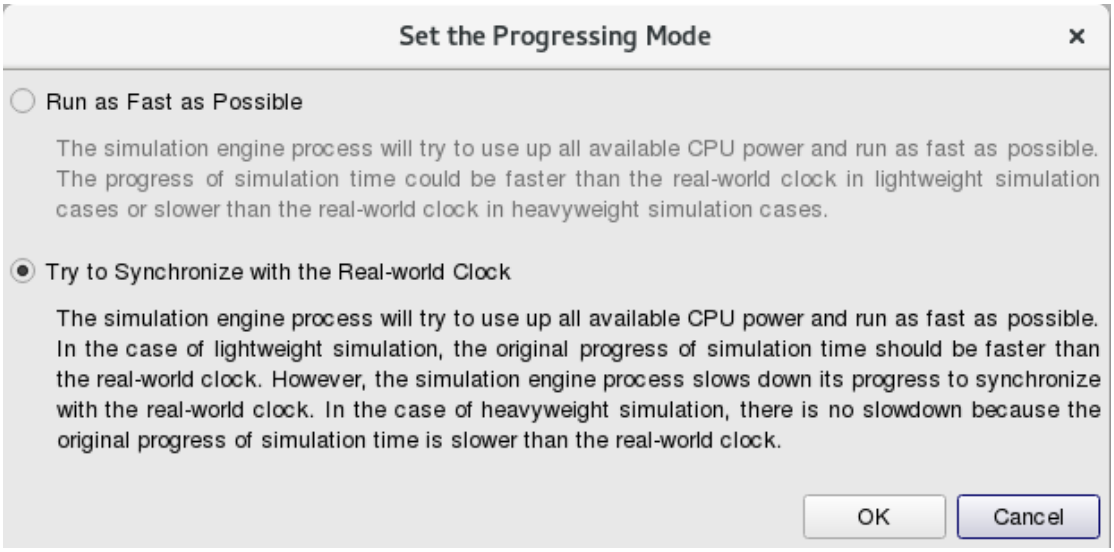
這個章節中同樣使用第一章中的 user_module01.xtpl 範例拓撲，並配合 user_module_01 的模組，本章中會介紹可以在模擬執行時，可以讀取或設定模組中變數的功能。

首先將範例拓撲中加入通訊流，首先將 HOST 1 中的 application 標籤加入 stcp -4 1.0.1.2，而 HOST2 的 application 標籤加入 rtcp -4，如下圖所示：

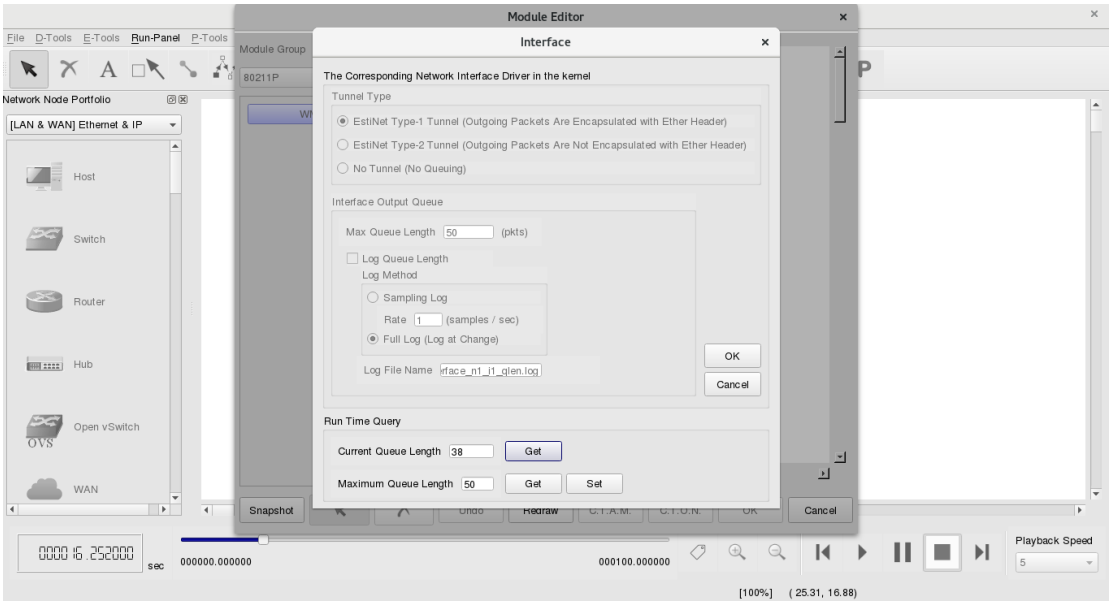

Host
Node ID
1
Node Type
Host
Application
Interface
Flow Classification
DNS
Routing
Firewall
Virtual Machine
Enable
Start (s)
Stop (s)
Command
Operation
2
100
stcp -4 1.0.1.2
C.T.O.N.
Add
Modify
Delete
Delete All
Enable All
Disable All
Adjust Start Time
Adjust Stop Time
App. Usage
Command Console
Module Editor
OK
C.P.T.O.N.
Cancel


Host
Node ID
2
Node Type
Host
Application
Interface
Flow Classification
DNS
Routing
Firewall
Virtual Machine
Enable
Start (s)
Stop (s)
Command
Operation
2
100
rtcp -4
C.T.O.N.
Add
Modify
Delete
Delete All
Enable All
Disable All
Adjust Start Time
Adjust Stop Time
App. Usage
Command Console
Module Editor
OK
C.P.T.O.N.
Cancel

接著為了不要讓模擬太快結束，以利觀察數值變化，點選上方標題列 E_Tools 的”Configure Simulation Processes→Simulation→Set the Progressing Mode→Try to Synchronize the Real-World Clock”，讓虛擬時間與真實時間一致。



接著切換到 G mode 執行模擬，執行模擬中在 Host1 上點選右鍵，點選 Module Editor，點選 Interface 模組兩下，看到最下方的 Export section 可以 Get 跟 Set 一些 Queue 參數，點選後可以從模組取到目前模擬中的數據，如下圖所示。



因此，在練習 2-1 中要在 User module01 中，加入這個 Run Time Query 的功能。

- 練習 2-1：修改 mdf

首先開啟 user module 01 的 mdf(開啟的位置跟方式在第一章說過，不再重複說明)，接著將最下方的 Export Section 改為下方紅字所示：

```
ModuleSection
    HeaderSection
        ModuleName          UserModule01
        GroupName           User_Defined
        Introduction        "This is a user-defined module."
        Parameter           myNumber 300 local
        Parameter           myString 111.111.111.111 local
    EndHeaderSection

    InitVariableSection
        Caption             "Parameters Setting"
        FrameSize           380 100

        Begin TEXTLINE      myNumber
            Caption         "My Number "
            Scale           10 18 220 30
            ActiveOn        MODE_EDIT
            Enabled         TRUE
```

```
            Type            INT
            Comment         "An Integer"
        End

        Begin TEXTLINE      myString
            Caption         "My String "
            Scale           10 48 220 30
            ActiveOn        MODE_EDIT
            Enabled         TRUE
            Type            IP
            Comment         "An IP string"
        End

        Begin BUTTON        b_ok
            Caption         "OK"
            Scale           250 17 60 30
            ActiveOn        ALL_MODE
            Action          ok
            Comment         "OK Button"
        End

        Begin BUTTON        b_cancel
            Caption         "Cancel"
            Scale           250 49 60 30
            ActiveOn        ALL_MODE
            Action          cancel
            Comment         "Cancel Button"
        End
EndInitVariableSection

ExportSection
        Caption             ""
        FrameSize           380 120

        Begin TEXTLINE      text_query_mynum
            Caption         "My Number "
            Scale           10 15 200 35
            ActiveOn        MODE_SIMULATION
```

```
        Enabled         TRUE
        Type            INT
        Comment         ""
End

Begin TEXTLINE          text_query_mystr
        Caption         "My String "
        Scale           10 55 200 35
        ActiveOn        MODE_SIMULATION
        Enabled         TRUE

        Type            INT
        Comment         ""
End

Begin ACCESSBUTTON      ab_get_mynum
        Caption         "Get"
        Scale           215 20 70 25
        ActiveOn        MODE_SIMULATION
        Enabled         TRUE

        Action          GET
        ActionObj       "export-my-number"

        Reference       text_query_mynum
        Comment         "get"
End

Begin ACCESSBUTTON      ab_get_mystr
        Caption         "Get"
        Scale           215 55 70 25
        ActiveOn        MODE_SIMULATION
        Enabled         TRUE

        Action          GET
        ActionObj       "export-my-string"

        Reference       text_query_mystr
```

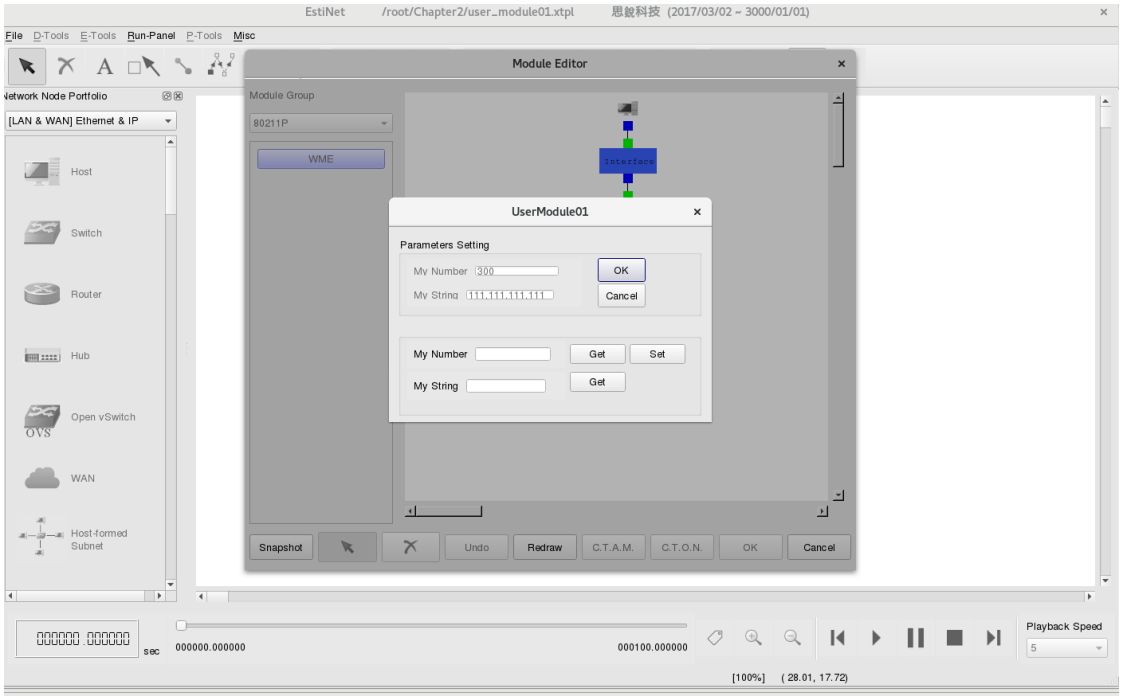
```
            Comment         "get"
        End

        Begin ACCESSBUTTON  ab_set_mynum
            Caption         "Set"
            Scale           290 20 70 25
            ActiveOn        MODE_SIMULATION
            Enabled         TRUE

            Action           SET
            ActionObj       "export-my-number"

            Reference       text_query_mynum
            Comment         "set"
        End
    EndExportSection
EndModuleSection
```

最下面 EXPORT SECTION 區塊中，將 CAPTION 加入”RUN TIME QUERY”，FRMAESIZE 把 0 0 改為 380 120(寬度、高度)，接著加入三個 ACCESSBUTTON 的物件，其中兩個為 ab_get_mynum 跟 ab_get_mystr 另一個為 ab_set_mystr。

修改後，在原始碼目錄中使用 make install，將 mdf 安裝到 GUI，再開啟 GUI 觀看修改後的 Layout。

Export section 的參數細節，使用者可以於附錄 C 查看。

改完之後的物件可以在開啟 GUI 後，在 G MODE，開啟 Host1 的 Module Editor 如下圖顯示：



- 修改 user_module_01.cc 檔：

接著，再回到原始碼目錄修改 user_module_01.cc，除了在第一個範例需要使用到 VBIND 等 API 外，還需要使用 EXPORT 的 API 以及在 command 函數增加解析 GUI 傳送過來的命令的程式碼，如下所示：

在 init()函數中，加入 EXPORT 的 API，EXPORT API 的第一個參數是 GUI 的 MDF ActionObj 所設定的字串，第二個參數則是設定此字串讀寫的權限，其參數值有 E_RONLY(只能讀不能寫)、E_WONLY (只能寫不能讀)以及 E_RONLY|E_E_WONLY(可讀寫) ，已"export-my-number"為例，此 EXPORT 如同下列紅字所示:

```
EXPORT("export-my-number", E_RONLY|E_WONLY);
```

接著修改 command 函式，這個函式每個模組中都有，是在 run time 時，可以讀取 GUI 傳送過來的命令。因此接收到 GUI 傳送過來的命令時，首先需判斷命令是屬於 GET 或 SET 哪種類型，接著再根據 ActionObj 所設定的字串取得或設定其對應的參數，如下方 code 紅色字體所示:

```
#include <stdlib.h>
#include <estinet_api.h>
#include <module/user-defined/user_module_01.h>
```

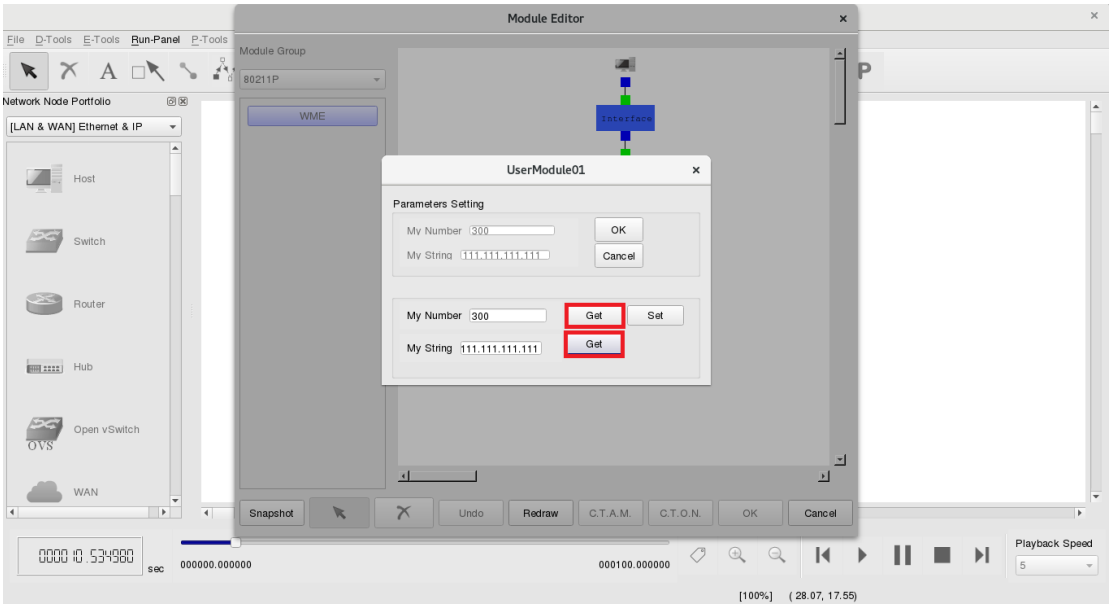
```
MODULE_GENERATOR(UserModule01);

UserModule01::UserModule01(u_int32_t type, u_int32_t id, struct plist* pl, const char *name)
        : NslObject(type, id, pl, name) {
        vBind_int("myNumber", &Number);
        vBind_char_str("myString", &String);
}

UserModule01::~UserModule01(){}

int UserModule01::init() {
        EXPORT("export-my-number", E_RONLY|E_WONLY);
        EXPORT("export-my-string", E_RONLY|E_WONLY);
        /* exercise 1 */
        printf("\e[1;31;40m\nExercise 1: Node ID = %d, myNumber = %d, myString = %s\e[m\n",
                        get_nid(), Number, String);
        return(NslObject::init());
}

int UserModule01::command(int argc, const char *argv[]) {

        char                tmpBuf[10];
        struct ExportStr    *ExpStr;
        u_int32_t           row,column;
        u_long *mytunidp;

        /* The Get implementation of Exported Variable "export-my-number" */
             if (!strcmp(argv[0], "Get")&&(argc==2)) {
                        if (!strcmp(argv[1], "export-my-number")) {
                        ExpStr = new ExportStr(1);
                        row = ExpStr->Add_row();
                        column = 1;

                        bzero(tmpBuf, sizeof(tmpBuf));
                                sprintf(tmpBuf, "%d", Number);
                        ExpStr->Insert_cell(row, column, tmpBuf, "\n");
                        EXPORT_GET_SUCCESS(ExpStr);
```

```
                        return 1;
                    }
            }

        /* The Get implementation of Exported Variable "export-my-string" */
            if (!strcmp(argv[0], "Get")&&(argc==2)) {
                    if (!strcmp(argv[1], "export-my-string")) {
                ExpStr = new ExportStr(1);
                row = ExpStr->Add_row();
                column = 1;

                bzero(tmpBuf, sizeof(tmpBuf));
                        sprintf(tmpBuf, "%s", String);
                ExpStr->Insert_cell(row, column, tmpBuf, "\n");
                EXPORT_GET_SUCCESS(ExpStr);
                        return 1;
                    }
            }

        /* The Set implementation of Exported Variable "export-my-number" */
            if (!strcmp(argv[0], "Set")&&(argc==3)) {
                    if (!strcmp(argv[1], "export-my-number")) {
                        Number = atoi(argv[2]);
                EXPORT_SET_SUCCESS();
                        return 1 ;
                    }
            }
        return(NslObject::command(argc, argv));
}

int UserModule01::recv(ePacket_ *pkt) {
        return(NslObject::recv(pkt));
}

int UserModule01::send(ePacket_ *pkt) {
        return(NslObject::send(pkt));
}
```

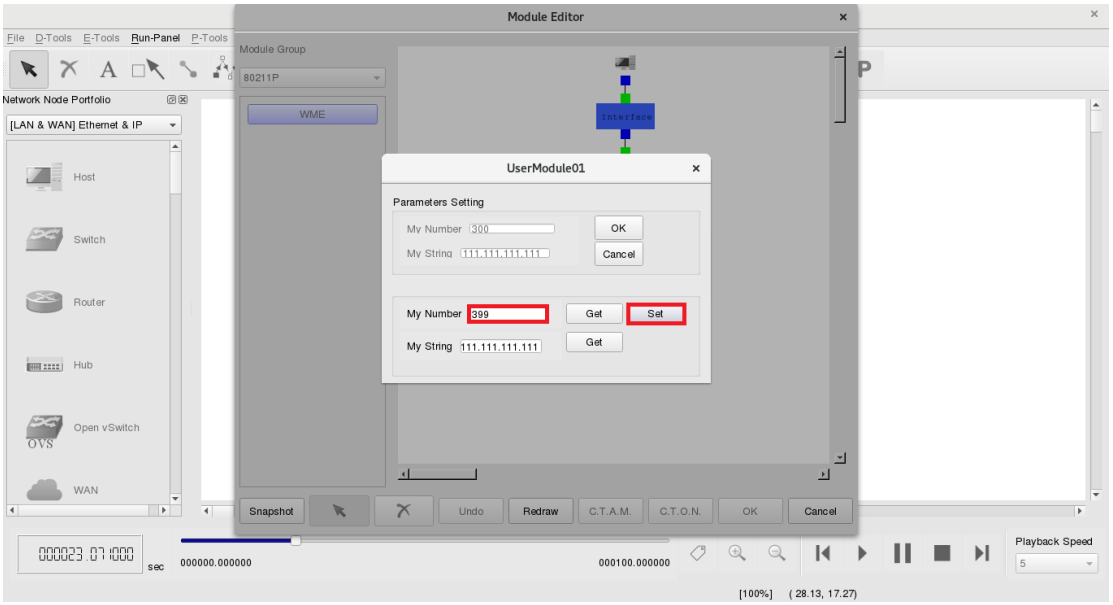
NOTE: VBIND 與 EXPORT 函數的差異
VBIND 是在模擬一開始時，讀取 if_and_medium_conf 中 MDF 已設定好的數值。 EXPORT 是在模擬中經由 IPC，在模組中 COMMAND 函數讀寫 GUI 中的 MDF 數值。每個模組中都有 COMMAND 函數。

完成後在原始碼目錄進行 make 與 make install。

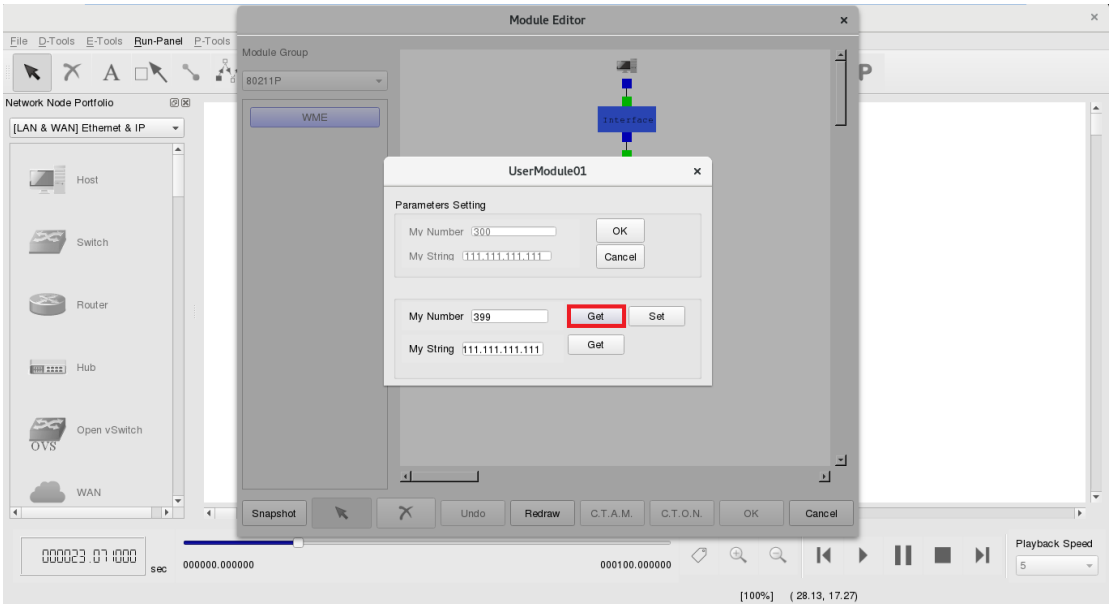
接著進行模擬，模擬時在 HOST1 上按右鍵選擇”Module Editor”按鈕，並在 user module01 點兩下，可以按下 Get 按鈕來取得目前模擬引擎中的 Number 的數值，也可以 Get 到模組中 String 的數值，如下圖所示：



接著修改 My Number 300 為 399，按下 Set 按鈕，如下圖所示：



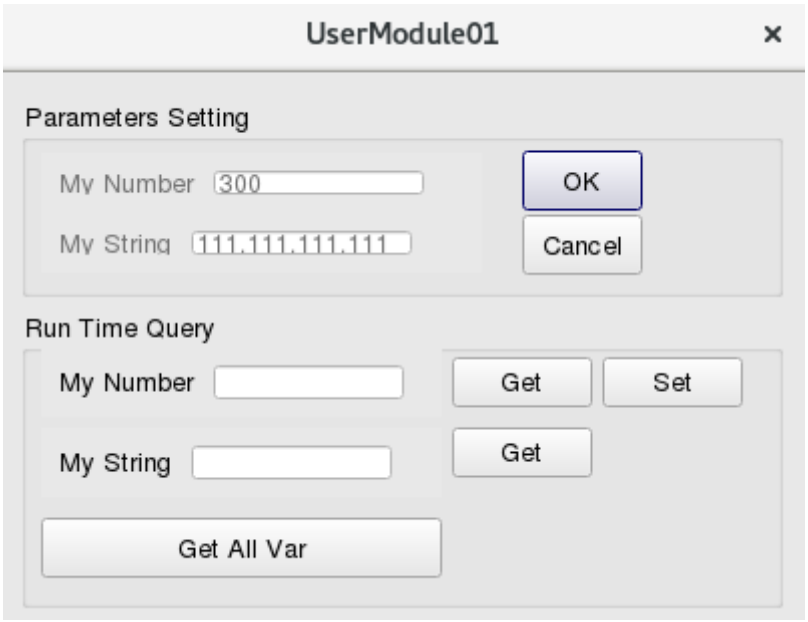
接著再按一次 My Number 的 Get 按鈕，發現 My Number 的值已經修改成功，如下圖所示：



- 另一種 EXPORT 物件
  RUN TIME QUERY 也可以用表格呈現的方式。

- 練習 2-2：

這種呈現方式是使用 GUI 中的 **INTERACTIONVIEW** 物件，因此修改 USER MODULE 01 的 MDF，改成如下圖所示：



將 EXPORT SECTION 的區塊改寫為如下所示：

```
ModuleSection
    HeaderSection
        ModuleName          UserModule01
        GroupName           User_Defined
        Introduction        "This is a user-defined module."
        Parameter           myNumber 300 local
        Parameter           myString 111.111.111.111 local
    EndHeaderSection

    InitVariableSection
        Caption             "Parameters Setting"
        FrameSize           380 100

        Begin TEXTLINE          myNumber
            Caption         "My Number "
            Scale           10 18 220 30
            ActiveOn        MODE_EDIT
            Enabled         TRUE
            Type            INT
            Comment         "An Integer"
        End

        Begin TEXTLINE          myString
            Caption         "My String "
            Scale           10 48 220 30
            ActiveOn        MODE_EDIT
            Enabled         TRUE
            Type            IP
            Comment         "An IP string"
        End

        Begin BUTTON        b_ok
            Caption         "OK"
            Scale           250 17 60 30
            ActiveOn        ALL_MODE
            Action          ok
            Comment         "OK Button"
        End
```

```
        Begin BUTTON        b_cancel
            Caption         "Cancel"
            Scale           250 49 60 30
            ActiveOn        ALL_MODE
            Action          cancel
            Comment         "Cancel Button"
        End
EndInitVariableSection

ExportSection
        Caption         "Run Time Query"
        FrameSize       380 150

        Begin TEXTLINE          text_query_mynum
            Caption         "My Number "
            Scale           10 10 200 35
            ActiveOn        MODE_SIMULATION
            Enabled         TRUE

            Type        INT
            Comment         ""
        End

        Begin TEXTLINE          text_query_mystr
            Caption         "My String "
            Scale           10 50 200 35
            ActiveOn        MODE_SIMULATION
            Enabled         TRUE

            Type        INT
            Comment         ""
        End

        Begin ACCESSBUTTON      ab_get_mynum
            Caption         "Get"
            Scale           215 15 70 25
            ActiveOn        MODE_SIMULATION
```

```
        Enabled         TRUE

        Action          GET
        ActionObj       "export-my-number"

        Reference       text_query_mynum
        Comment         "get"
End

Begin ACCESSBUTTON      ab_get_mystr
        Caption         "Get"
        Scale           215 50 70 25
        ActiveOn        MODE_SIMULATION
        Enabled         TRUE

        Action          GET
        ActionObj       "export-my-string"

        Reference       text_query_mystr
        Comment         "get"
End

Begin ACCESSBUTTON      ab_set_mynum
        Caption         "Set"
        Scale           290 15 70 25
        ActiveOn        MODE_SIMULATION
        Enabled         TRUE

        Action           SET
        ActionObj       "export-my-number"

        Reference       text_query_mynum
        Comment         "set"
End

Begin INTERACTIONVIEW  iv_get_all
        Caption         "Get All Var"
        Scale           10 100 200 30
```

```
                ActiveOn        MODE_SIMULATION
                Enabled         TRUE

                Action          GET
                ActionObj       "export-all-data"

                Fields          "My String" "My Number"
                Comment         "All Data"
            End

    EndExportSection
EndModuleSection
```

user_module01.cc 修改成：

```
#include <stdlib.h>
#include <estinet_api.h>
#include <module/user-defined/user_module_01.h>

MODULE_GENERATOR(UserModule01);

UserModule01::UserModule01(u_int32_t type, u_int32_t id, struct plist* pl, const char *name)
    : NslObject(type, id, pl, name) {
    vBind_int("myNumber", &Number);
    vBind_char_str("myString", &String);
}

UserModule01::~UserModule01(){}

int UserModule01::init() {
     EXPORT("export-my-number", E_RONLY|E_WONLY);
    EXPORT("export-my-string", E_RONLY|E_WONLY);
    EXPORT("export-all-data", E_RONLY|E_WONLY);
    /* exercise 1 */
    printf("\e[1;31;40m\nExercise 1: Node ID = %d, myNumber = %d, myString = %s\e[m\n",
            get_nid(), Number, String);
    return(NslObject::init());
}
```

```
int UserModule01::command(int argc, const char *argv[]) {

        char                tmpBuf[10];
        struct ExportStr    *ExpStr;
        u_int32_t           row,column;
        u_long *mytunidp;

        /* The Get implementation of Exported Variable "export-my-number" */
            if (!strcmp(argv[0], "Get")&&(argc==2)) {
                        if (!strcmp(argv[1], "export-my-number")) {
                    ExpStr = new ExportStr(1);
                    row = ExpStr->Add_row();
                    column = 1;

                    bzero(tmpBuf, sizeof(tmpBuf));
                                sprintf(tmpBuf, "%d", Number);
                    ExpStr->Insert_cell(row, column, tmpBuf, "\n");
                    EXPORT_GET_SUCCESS(ExpStr);
                                return 1;
                        }
            }

        /* The Get implementation of Exported Variable "export-my-string" */
            if (!strcmp(argv[0], "Get")&&(argc==2)) {
                        if (!strcmp(argv[1], "export-my-string")) {
                    ExpStr = new ExportStr(1);
                    row = ExpStr->Add_row();
                    column = 1;

                    bzero(tmpBuf, sizeof(tmpBuf));
                                sprintf(tmpBuf, "%s", String);
                    ExpStr->Insert_cell(row, column, tmpBuf, "\n");
                    EXPORT_GET_SUCCESS(ExpStr);
                                return 1;
                        }
            }
```

```
        /* The Set implementation of Exported Variable "export-my-number" */
            if (!strcmp(argv[0], "Set")&&(argc==3)) {
                        if (!strcmp(argv[1], "export-my-number")) {
                                    Number = atoi(argv[2]);
                        EXPORT_SET_SUCCESS();
                                    return 1 ;
                        }
             }

            /* The Set implementation of Exported Variable "export-all-data" */
            if (!strcmp(argv[0], "Get")&&(argc==2)) {
                        if (!strcmp(argv[1], "export-all-data")) {

                        ExpStr = new ExportStr(2);

                        bzero(tmpBuf, sizeof(tmpBuf));
                        sprintf(tmpBuf, "String\t\tNumber\n");
                        ExpStr->Insert_comment(tmpBuf);

                        row = ExpStr->Add_row();
                        column = 1;

                        bzero(tmpBuf, sizeof(tmpBuf));
                              sprintf(tmpBuf, "%s", String);
                        ExpStr->Insert_cell(row, column++, tmpBuf, "\t\t");

                        bzero(tmpBuf, sizeof(tmpBuf));
                        sprintf(tmpBuf, "%d", Number);
                        ExpStr->Insert_cell(row, column, tmpBuf, "\n");

                        EXPORT_GET_SUCCESS(ExpStr);
                                    return 1 ;
                        }
             }

        return(NslObject::command(argc, argv));
}

int UserModule01::recv(ePacket_ *pkt) {
```

```
        return(NslObject::recv(pkt));
}

int UserModule01::send(ePacket_ *pkt) {
        return(NslObject::send(pkt));
}
```

修改完畢後，在原始碼目錄中執行 make、make install，並執行模擬，即可看到上圖中的呈現效果。

第三章：模組間互相分享資料

本章重點：
(1)、介紹 REG_VAR、GET_REG_VAR 的使用範例
(2)、介紹 INSTANCELOOKUP 的使用範例

下載練習：


Chapter3.tar.bz2

這一章說明模組之間分享變數或是函式給其它模組使用，例如模組中若需要取得 phy 模組的資料來運算(如頻寬資料等)。在模擬器的模組平台運作中，每個節點上都有自己的 protocol stack，因此會有許多的模組運作，在節點結構中提供一個公用的資料結構為”var-register-table”，可以讓 protocol stack 中的模組，註冊變數到此結構中，而其它模組就可以到這個資料結構來取得此分享的變數資料。

- 註冊變數
  REG_VAR()是註冊模組中的變數到節點公用資料結構--"var-register-table"，讓這個節點上 protocol stack 的所有模組，都可以讀取所註冊的變數。
  例如在本練習範例中，會在 phy 模組中取得 USER MODULE 01 所註冊的變數。
  用法：REG_VAR(vname, var)
  第一個參數是一個指標，指向一個識別的名稱，這個名稱將用來識別這些註冊的變數；而第二個參數則是一個指標指向這個分享的變數。

- 取得其它模組中的註冊變數
  GET_REG_VAR()是取得這個節點中的其它模組所註冊的變數。
  用法：GET_REG_VAR(interface id, vname, type)
  第一個參數是 interface id，是指 protocol stack 上的 interface。我們可以使用 API(get_ifid())來取得 protocol stack 上的 interface id，更精準的指向某個模組所分享出來的變數。

- get_ifid()用法

get_ifid()會回傳目前模組所連接到的 interface id。

■ 練習 3-1：REG_VAR()與 GET_REG_VAR()

首先使用第一章中的範例拓撲，並配合 USER MODULE 01 的模組，修改 user_module01.cc 的檔案。將 Number 變數分享給其它模組使用，在建構子中加入註冊的資訊，如下方紅字所示：

user_module01.cc 修改成：

```
#include <stdlib.h>
#include <estinet_api.h>
#include <module/user-defined/user_module_01.h>

MODULE_GENERATOR(UserModule01);

UserModule01::UserModule01(u_int32_t type, u_int32_t id, struct iflist* ifl, const char
*name): NslObject(type, id, ifl, name) {
        vBind_int("myNumber", &Number);
        vBind_char_str("myString", &String);
        REG_VAR("shared-number", &Number);
}

UserModule01::~UserModule01(){}

int UserModule01::init() {
        return(NslObject::init());
}

int UserModule01::command(int argc, const char *argv[]) {
        return(NslObject::command(argc, argv));
}

int UserModule01::recv(ePacket_ *pkt) {
        return(NslObject::recv(pkt));
}

int UserModule01::send(ePacket_ *pkt) {
        return(NslObject::send(pkt));
}
```

接著，在 protocol stack 的 PHY 模組，取得在 user_module01 模組中所分享出來變數值。修改原始碼目錄中的 module/8023/phy/phy.cc，主要只在 init()函數中，列印 user_module01 模組所分享出來的變數值，如下方紅字。

```
int phy::init() {
        NslObject::init();
        int *get_share_data = GET_REG_VAR(get_ifid(), "shared-number", int *);
        if (get_share_data)
        {
                printf("\e[1;36;46m\nExercise3-1: Get Shared Number in UserModule01=
%d\e[m\n", *get_share_data);
        }
 }
```

執行結果，如下圖所示，phy 模組中，可以經由 **GET_REG_VAR** 跟取得 UserModule01 中註冊的變數。

```
e's initial location and mobility events (if any) that will be triggered during
simulation.
add_interface: add interface, node:1 port_no:1 dev:tun1 rename:eth0
[FlowClassifier] Warning: No flow rule setting file /root/.estinet/estinetss/wor
kdir/1525186664-job/per_node/node1/user_module01.n1_i1.flow_rule, use default pr
iority.

Exercise3-1: Get Shared Number in UserModule01= 300
In Node::init(), the initialization of node 1's all modules succeeds.
add_interface: add interface, node:2 port_no:1 dev:tun2 rename:eth0
[FlowClassifier] Warning: No flow rule setting file /root/.estinet/estinetss/wor
kdir/1525186664-job/per_node/node2/user_module01.n1_i1.flow_rule, use default pr
iority.

Exercise3-1: Get Shared Number in UserModule01= 300
In Node::init(), the initialization of node 2's all modules succeeds.
RanSeed=485297
=====================================================
=============== Start executing events ===============
=====================================================
current ticks= 0, run "2 sh init.sh"
```

■ 練習 3-2：REG_VAR()與 GET_REG_VAR()

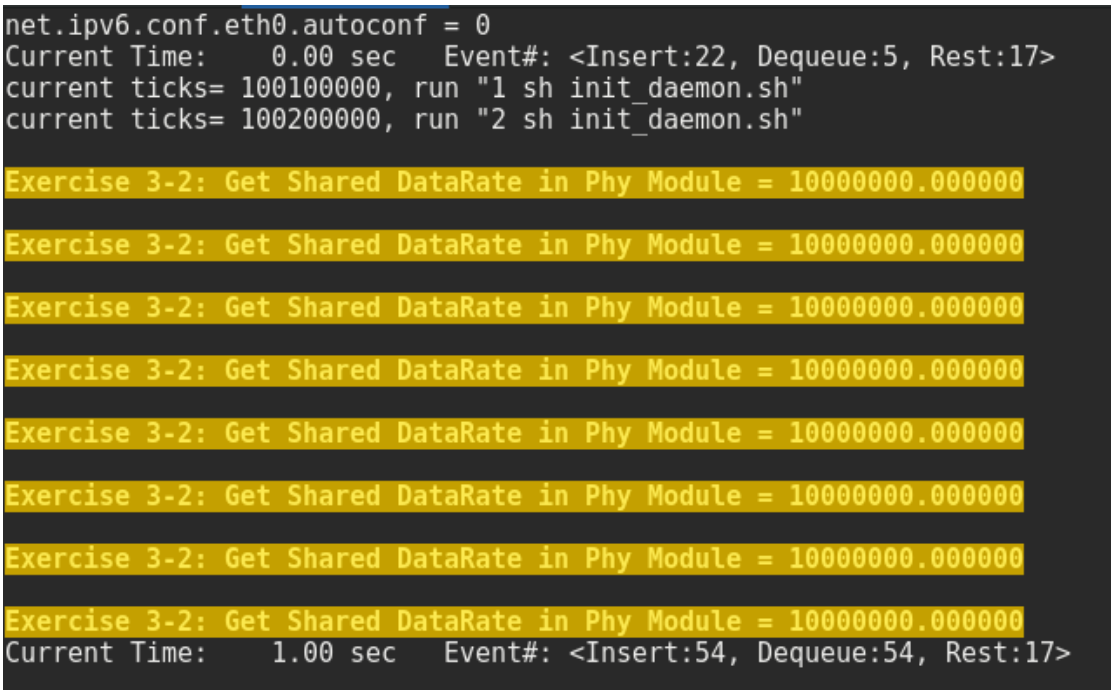
在 phy 模組的程式中，可以看到也有註冊分享給其它模組的變數如下：

```
REG_VAR("DataRate", &bw_);
```

這個變數即為頻寬的資料，因此練習在 UserModule01 中 send 函式中，取得 phy 模組的 DataRate 資料。

參考解答：

```
int UserModule01::send(ePacket_ *pkt) {

printf("\e[1;33;43m\nExercise 3-2: Get Shared DataRate in Phy Module = %f\e[m\n",
*(GET_REG_VAR(get_ifid(), "DataRate", double *)));
    return(NslObject::send(pkt));
}
```



註：在 user_module01 模組的 init 函式執行上述紅字的話，取得的資料會是 0，因為 phy 的 init 的執行比 user_module01 的 init 執行較晚。因此選擇 send()函數中列印(但記得要有通訊流，如 ipv6 廣播封包或 stcp/rtcp，才會執行到 send 函式)

另外，還有一種方式是模組可以使用其它模組的函式，這個做法需要使用到 InstanceLookup 這個 API。

- InstanceLookup 的用法

首先第一個參數要放這個節點的 ID，可以用 get_nid()這個 API 來取得目前該節點的 ID；第二個參數則是使用 protocol stack 目前連接的 interface id，可以使用 get_ifid()來取得；接著第三個參數則是放 phy 模組註冊在整個模擬引擎中的名字，可以透過此名稱的指到此物件，後面就可以依照 c++的用法，接成員變數及函式，如下例：

```
((phy *)InstanceLookup(get_nid(), get_ifid(), "Phy"))->Debugger();
```

- 練習 3-3：InstanceLookup()函數

首先改寫 user_module01.cc 的程式碼，加入下面紅字的程式碼，將 phy.h include 進來，主要等一下要用到 phy 物件中的函式

接著在 send()函式中，使用 InstanceLookup 這個 API，來執行 phy 物件中的 Debugger 函式。

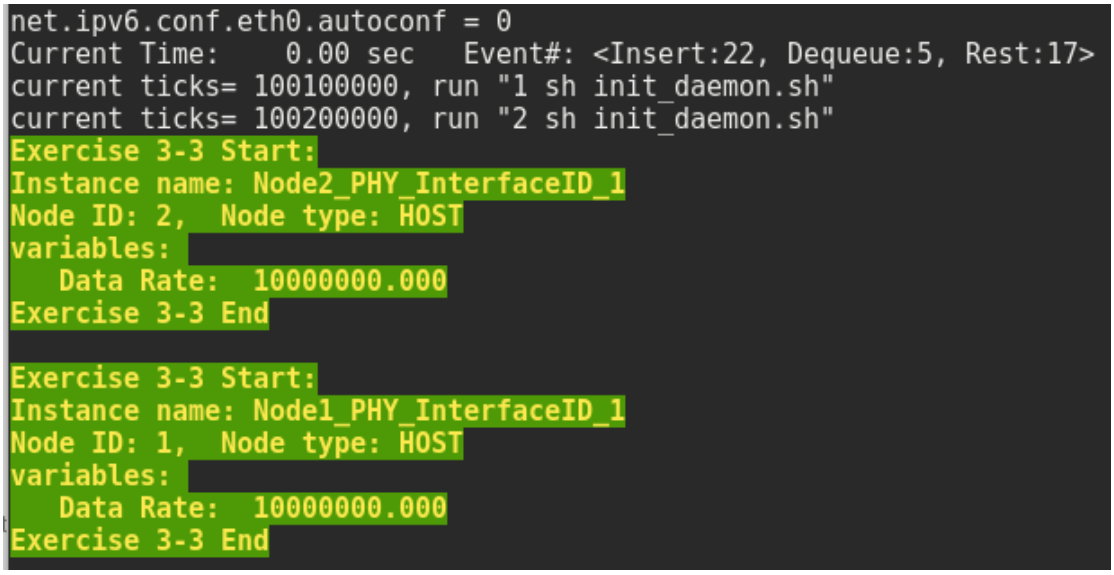
```
#include <stdlib.h>
#include <estinet_api.h>
#include <module/user-defined/user_module_01.h>
#include <module/8023/phy/phy.h>

MODULE_GENERATOR(UserModule01);

UserModule01::UserModule01(u_int32_t type, u_int32_t id, struct iflist* ifl, const char
*name): NslObject(type, id, ifl, name) {
        vBind_int("myNumber", &Number);
        vBind_char_str("myString", &String);
        REG_VAR("shared-number", &Number);
}

UserModule01::~UserModule01(){}

int UserModule01::init() {
        return(NslObject::init());
}
```

```
int UserModule01::command(int argc, const char *argv[]) {
        return(NslObject::command(argc, argv));
}

int UserModule01::recv(ePacket_ *pkt) {
        return(NslObject::recv(pkt));
}

int UserModule01::send(ePacket_ *pkt) {
        ((phy *)InstanceLookup(get_nid(), get_ifid(), "PHY"))->Debugger();
        return(NslObject::send(pkt));
}
```

接著為了讓 PHY 模組中的 Debugger()函式列印到終端機上比較明顯，將此函式加入 ascii code 的顯示效果：

```
int phy::Debugger() {
        printf("\e[1;33;42mExercise 3-3 Start:\n");
        NslObject::Debugger();
        printf("      Data Rate: %13.3lf\n", bw_);
        printf("Exercise 3-3 End\e[m\n\n");
        return(1);
}
```

make 以及 make install 後，並執行模擬，可以看到如下圖綠色區塊的效果



因此可以透過此 API，來執行其它模組中的函式。

第四章：RUN TIME MESSAGE

本章重點：
(1)、介紹 RUN TIME MESSAGE -- WARNING 的使用範例 1
(2)、介紹 RUN TIME MESSAGE -- INFOMATION 的使用範例 2
(3)、介紹 RUN TIME MESSAGE – FATAL ERROR 的使用範例 3
(4)、模組的 send()與 recv()函式架構

下載練習：


Chapter4.tar.bz2

Run Time Message 可以在原始碼中預先安排一些錯誤提示、警告提示、或是一些資訊提示的程式碼，使得模擬進行時只要執行到這些程式碼，會將這些訊息經由 IPC 送到 GUI，讓 GUI 跳出視窗與使用者進行互動。

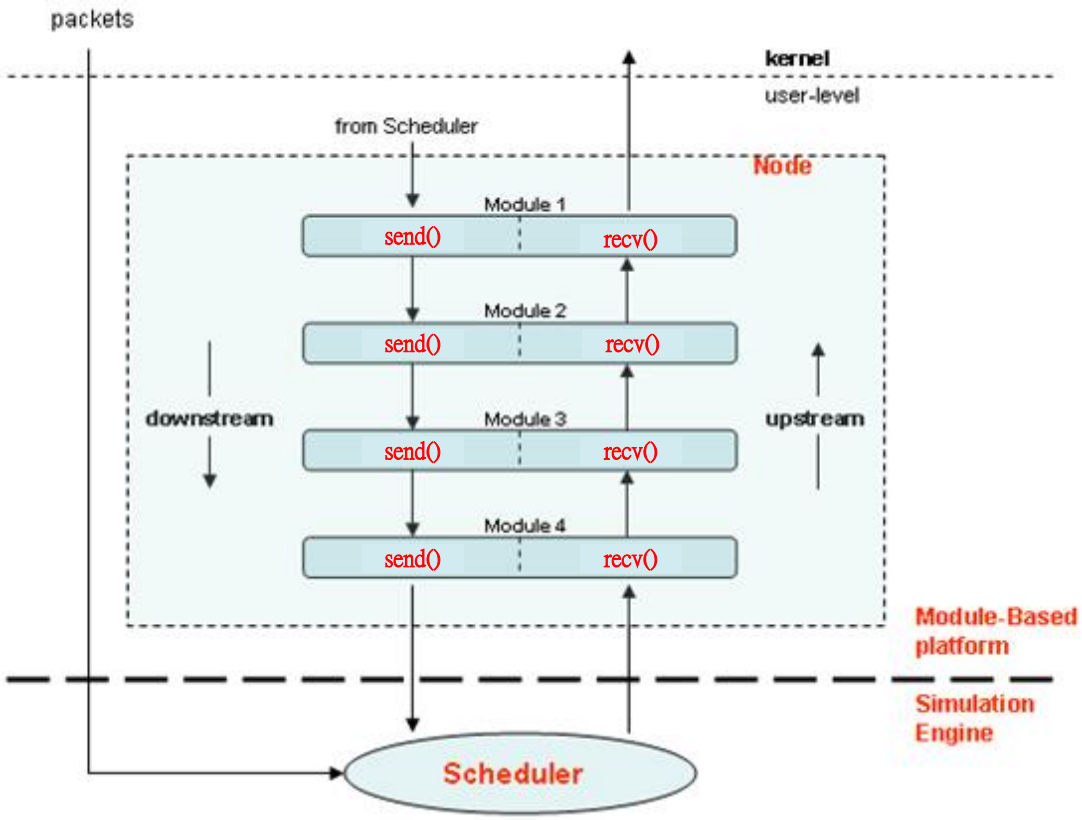
Run Time Message 在模擬引擎中分為三種：

- INFORMATION
- WARNING
- FATAL ERROR

使用三個範例程式碼來展示。

使用第一章中的範例拓撲(user_module01.xtpl)。
預期都在 user_module01.cc 的 send()函式中，各加入一行程式碼展示。

註：由於封包進入模組時，會由 send()進入，處理完成後再送給下一個模組的 send()，直到送到最底層的模組，才會傳送給其它節點。接收時則是由每個模組的 recv()由最底層的模組，逐步往上送給每一個模組處理，最後才由 interface 模組將封包經由 kernel 再導向應用程式。如下圖所示：



因此使用通訊流來觸發 user_module01 模組中的 send()函式，並使用下面三種 run time message 種類來展示：

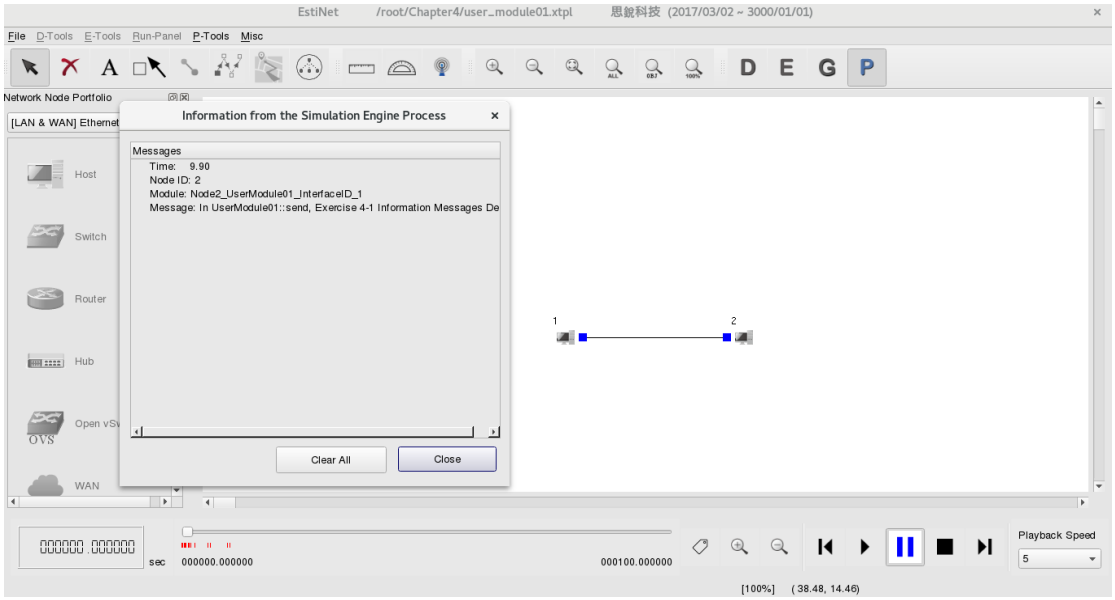
- 練習 4-1：INFORMATION

  sendRuntimeMsg 這個 API 是用來展示 RUN TIME MESSAGE 的 API，第一個參數是指 RUN TIME MESSAGE 的 TYPE，在本章的練習 4-1 中，使用”**RTMSG_INFORMATION**”的 TYPE，第二個參數是放 Node id，第三個參數是模組的名稱，可以用 get_name()這個 API 取得，接著第四個參數是我們要呈現在 GUI 視窗中的訊息。如下紅字所示：

```
int UserModule01::send(ePacket_ *pkt) {
        sendRuntimeMsg(RTMSG_INFORMATION, get_nid(), get_name(), "In UserModule01::send,
Exercise 4-1 Information Messages Demo");
        return(NslObject::send(pkt));
}
```

make 後再執行 make install，完成編譯以及安裝。接著執行模擬，就會看到有一個視窗不斷的印出訊息，這是因為通訊流不斷的有資料，每一筆封包進入 send

函式，就會觸發一次這個功能，GUI 會將所有的 INFORMATION 集中在同一個視窗中。



■ 練習 4-2：WARNING

接下來繼續修改 user_module01.cc 的程式碼。將練習 4-1 中 sendRuntimeMsg 的第一個參數 TYPE 改為”**RTMSG_WARNING**”，並將第四個參數中的訊息加以改變，如下紅字所示：

```
int UserModule01::send(ePacket_ *pkt) {
        sendRuntimeMsg(RTMSG_WARNING, get_nid(), get_name(), "In UserModule01::send,
Exercise 4-2 Warning Messages Demo");
        return(NslObject::send(pkt));
}
```

make 後再執行 make install，完成編譯以及安裝。接著執行模擬時，可以看到模擬會暫停，並跳出一個互動視窗，詢問使用者是否要”Continue”或是”Stop”。如果按”Stop”會直接結束模擬，或是按”Continue”的話，則因為下一個封包馬上又觸發這個 WARNING 的訊息。使用者可以決定此類訊息所出現的時機。

■ 練習 4-3：FATAL ERROR

接下來繼續修改 user_module01.cc 的 send 函式，sendRuntimeMsg 第一個參數的 TYPE 為 **RTMSG_FATAL_ERROR**，並將第四個參數中的訊息稍加修改，如下紅字所示：

```
int UserModule01::send(ePacket_ *pkt) {
        sendRuntimeMsg(RTMSG_FATAL_ERROR, get_nid(), get_name(), "In
UserModule01::send, Exercise 4-3 Error Messages Demo");
    return(NslObject::send(pkt));
}
```

make 後再執行 make install，完成編譯以及安裝。接著執行模擬只出現一個互動選項，要求使用者只能結束模擬，這在遇到一些錯誤情況無法再進行模擬時，開發者可以運用此訊息要求結束模擬。

這三種 RUN TIME MESSAGE 都可以設定，可讓使用者經由 GUI 知道與模擬引擎的互動資訊。三種 RUN TIME MESSAGE 的用途不太相同，開發者可以依自身需求決定使用哪一種 TYPE 的 RUN TIME MESSAGE。

第五章：不需要使用 GUI 的模擬執行方式

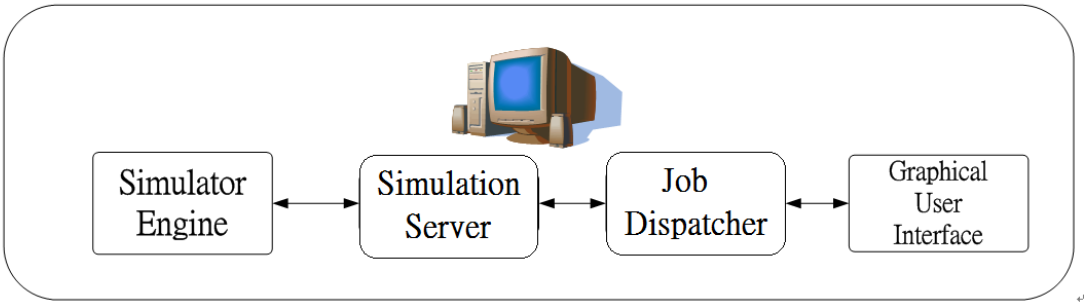
本章重點：
(1)、開發的架構
(2)、關掉 IPC 的使用方式
(3)、查看 Frame Trace File

下載練習：

Chapter5.tar.bz2

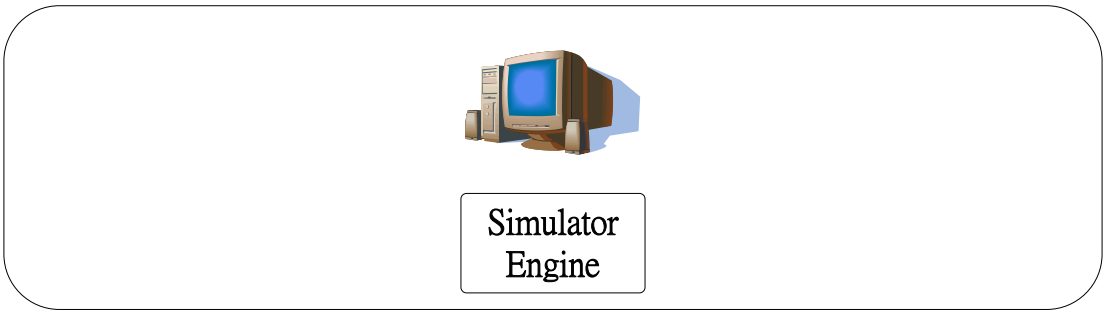
在一些開發中的情況，開發者希望一個更單純的執行環境，希望跟 GUI 之間的 IPC 關閉，純粹只有模擬引擎的執行方式。因此本章中將說明如何在沒有 GUI 的情況下執行模擬。

一般常用的單機架構中，會像下圖架構：(ESTINET 也是分散式架構，於附錄 D 說明)



如同在第一章中，在同一台電腦中執行 estinetjd、estinetss、estinetgui 來執行模擬。GUI 負責倒出模擬的設定檔後，將這些設定檔送給 estinetjd 來分配工作，而 estinetjd 再透過 estinetss 來找到閒置的模擬引擎來讀取這些設定檔來進行模擬。但這樣在開發過程中，有時模組開發中，不見得 GUI 都有支援開發中的節點類型，或是開發人員希望有一個更單純的模擬環境時，開發者可以選擇使用關掉 IPC，只留下模擬引擎存在。

若關掉 IPC 的情況下，架構會更單純如下圖：



開發人員常在這樣的情況下，設計 Protocol Module 或 Debug，在開發模組及協定中，更為方便。

因此，使用原本的 user_ moduler01 拓撲(user_moduler01.xtpl)做為說明。

一般來說，模擬使用 GUI 執行過後會有三個目錄以及跟一個 GUI 用的檔案：user_ moduler01.xtpl 以及 user_ moduler01.gui_data 目錄是 GUI 在儲存一些 GUI 物件資訊的資料跟資料目錄，是為 GUI 所使用。而 user_moduler01.sim 目錄則是會送交一些設定檔(例如 if_and_medium_conf 檔)給模擬引擎，而 user_moduler01.results 目錄是模擬引擎執行的執行結果。

關鍵步驟就是要將 sim 目錄中的設定檔送給模擬引擎。

在不需要使用 GUI 來執行模擬的步驟如下：
首先，要設定一個環境變數”ESTINET_WORKDIR”，設定此目錄到 sim 目錄，如下圖所示：

export ESTINET_WORKDIR=/root/Chapter5/user_module01.sim/

```
[root@localhost Chapter5]# ls user_module01.sim/.for_se_direct_access/
per_node                                                    user_module01.obstacle
runtime_fatal_error_message_log   user_module01.if_and_medium_conf   user_module01.run
user_module01.application                                   user_module01.traffic_light_log
user_module01.car_moving_path_log  user_module01.node_type_and_virtualization
[root@localhost Chapter5]# export ESTINET_WORKDIR=/root/Chapter5/user_module01.sim/
[root@localhost Chapter5]# estinetse -d $ESTINET_WORKDIR/.for_se_direct_access/user_module01.if_and_medium_conf
docker daemon is running.
docker images: estinet10/fedora24:v1
turn off docker seccomp setting
clean old container
The maximum number of kernel-supported tunnel interfaces is 40960.
The maximum queue length of kernel-supported data tunnel interfaces is 1000.
The maximum queue length of kernel-supported event tunnel interfaces is 50000.
The limited maximum number of file descriptors is 1048576.
The limited maximum number of forked processes is unlimited.
The limited maximum size of a core dump file is unlimited.
In HeapObject::HeapObject(), the event heap's capacity is 5000000.
Trying to connect to license server 1, please wait
LogID : 36998
Get capablilty_v2 command
```

接著執行 estinetse 下-d 參數表示，關閉與 GUI 之間的 IPC，後面再接上 if_and_medium_conf 的檔案路徑：
$ESTINET_WORKDIR/.for_se_direct_access/user_moduler01. if_and_medium_conf 即可。

estinetse -d $ESTINET_WORKDIR/.for_se_direct_access/user_module01.if_and_medium_conf

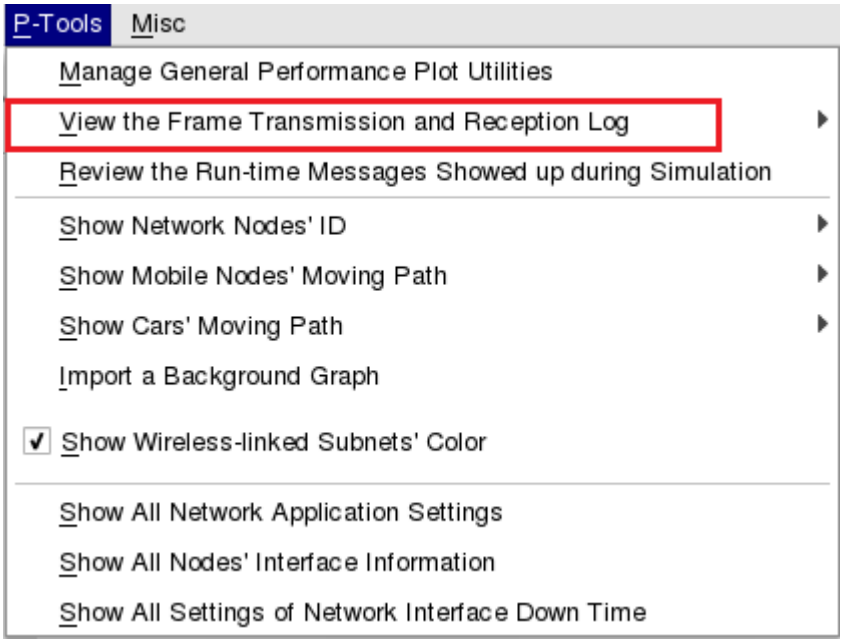
按下 enter 後，開始執行模擬。

```
Current Time:    3.00 sec   Event#: <Insert:2318, Dequeue:2317, Rest:21>
6664     3          1182 Kbyte/sec ==>  9.460160 Mbit/sec
Current Time:    4.00 sec   Event#: <Insert:2299, Dequeue:2299, Rest:21>
6664     4          1183 Kbyte/sec ==>  9.464128 Mbit/sec
Current Time:    5.00 sec   Event#: <Insert:2299, Dequeue:2299, Rest:21>
6664     5          1183 Kbyte/sec ==>  9.464128 Mbit/sec
Current Time:    6.00 sec   Event#: <Insert:2301, Dequeue:2301, Rest:21>
6664     6          1183 Kbyte/sec ==>  9.464128 Mbit/sec
Current Time:    7.00 sec   Event#: <Insert:2300, Dequeue:2300, Rest:21>
6664     7          1183 Kbyte/sec ==>  9.464128 Mbit/sec
Current Time:    8.00 sec   Event#: <Insert:2299, Dequeue:2299, Rest:21>
6664     8          1183 Kbyte/sec ==>  9.464128 Mbit/sec
Current Time:    9.00 sec   Event#: <Insert:2299, Dequeue:2299, Rest:21>
6664     9          1183 Kbyte/sec ==>  9.464128 Mbit/sec
Current Time:   10.00 sec   Event#: <Insert:2300, Dequeue:2300, Rest:21>
6664     10         1183 Kbyte/sec ==>  9.464128 Mbit/sec
Current Time:   11.00 sec   Event#: <Insert:2300, Dequeue:2300, Rest:21>
6664     11         1183 Kbyte/sec ==>  9.464128 Mbit/sec
Current Time:   12.00 sec   Event#: <Insert:2298, Dequeue:2298, Rest:21>
6664     12         1183 Kbyte/sec ==>  9.464128 Mbit/sec
```

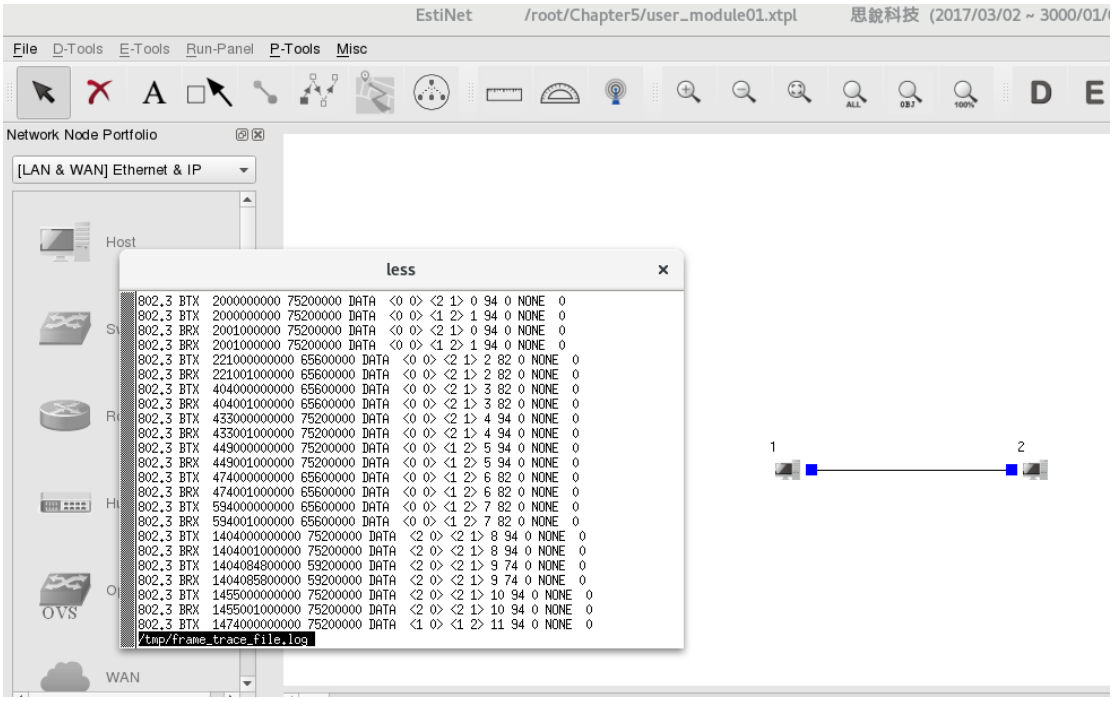
模擬結束後，會在$ESTINET_WORKDIR 中，存下模擬結果 frame_trace_file 檔案，如下圖所示，ESTINET_WORKDIR 在 sim 目錄中，因此 frame_trace_file 檔也就存在這裡的.for_se_direct_access 了(平常透過 GUI 會存在 results 目錄中，但關掉 ipc 的方式會存在 ESTINET_WORKDIR 中)。
所謂的 frame_trace_file 檔，在模擬器中，主要是記錄 mac 層中的封包的活動記錄。

```
[root@localhost user_module01.sim]# ls
application_setting          networking_setting  user_module01.node_type_and_virtualization
interface_and_medium_setting  particular_field    user_module01.run
[root@localhost user_module01.sim]# cd .for_se_direct_access/
[root@localhost .for_se_direct_access]# ls
per_node                                user_module01.frame_trace_file          user_module01.obstacle
runtime_fatal_error_message_log         user_module01.if_and_medium_conf        user_module01.run
user_module01.application               user_module01.moving_path               user_module01.traffic_light_log
user_module01.car_moving_path_log       user_module01.node_type_and_virtualization
```

frame_trace_file 檔在 GUI 中，可以用 view frame trace file 的方式來讀取，如下圖所示，按上面標題列的 P_Tools→View the Frame Transmission and Reception Log→Open the Frame Trace File，選取 frame_trace_file 檔案

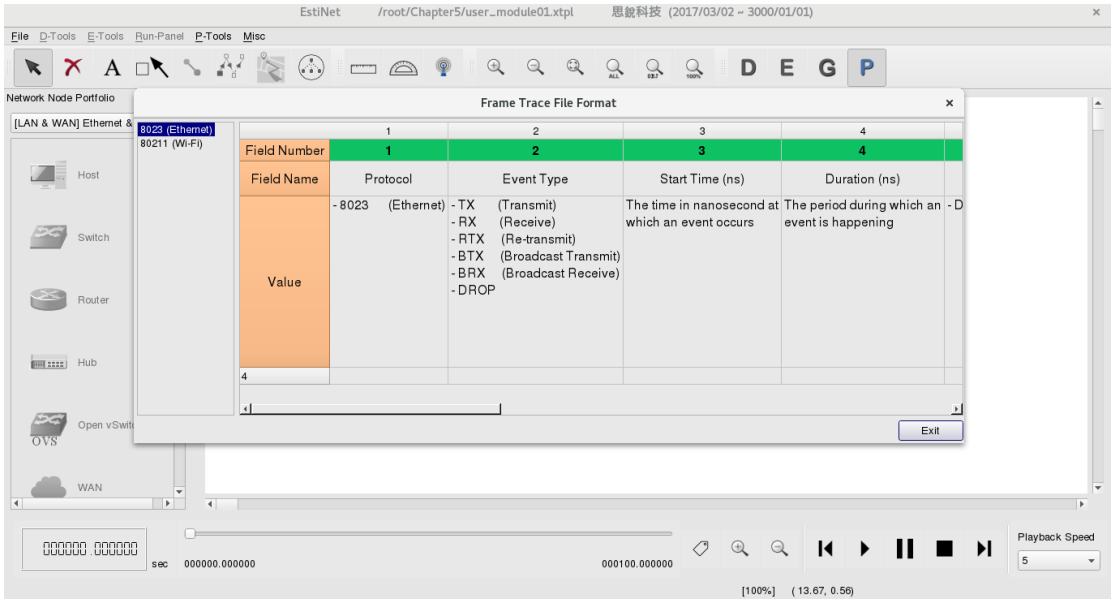


就會看到 frame_trace_file 的資料



至於 frame_trace_file 上每一行中的每一欄所代表的意義，可以看 GUI 中，按上面標題列的 P_Tools→View the Frame Transmission and Reception Log/Show the Frame Trace File's Format

按下後可以看到每一欄所代表的意義。



但如果在不想開 GUI 的情況下看 frame_trace_file 檔，也可以使用 estinet_printftr 工具來檢視或匯出。

estinet_printftr /root/Chapter5/user_module01.sim/.for_se_direct_access/user_module01.frame_trace_file

```
[root@localhost Chapter5]# estinet_printftr /root/Chapter5/user_module01.sim/.for_se_direct_access/user_module01.frame_trac
e_file
```

```
802.3 TX  99991082400000 1214400000 DATA  <1 2> <1 2> 120132 1518 0 NONE  0
802.3 RX  99991083400000 1214400000 DATA  <1 2> <1 2> 120132 1518 0 NONE  0
802.3 TX  99992306400000 1214400000 DATA  <1 2> <1 2> 120133 1518 0 NONE  0
802.3 RX  99992307400000 1214400000 DATA  <1 2> <1 2> 120133 1518 0 NONE  0
802.3 TX  99993521800000 56000000 DATA  <2 1> <2 1> 120191 70 0 NONE  0
802.3 RX  99993522800000 56000000 DATA  <2 1> <2 1> 120191 70 0 NONE  0
802.3 TX  99993530400000 1214400000 DATA  <1 2> <1 2> 120134 1518 0 NONE  0
802.3 RX  99993531400000 1214400000 DATA  <1 2> <1 2> 120134 1518 0 NONE  0
802.3 TX  99994754400000 1214400000 DATA  <1 2> <1 2> 120135 1518 0 NONE  0
802.3 RX  99994755400000 1214400000 DATA  <1 2> <1 2> 120135 1518 0 NONE  0
802.3 TX  99995969800000 56000000 DATA  <2 1> <2 1> 120192 70 0 NONE  0
802.3 RX  99995970800000 56000000 DATA  <2 1> <2 1> 120192 70 0 NONE  0
802.3 TX  99995978400000 1214400000 DATA  <1 2> <1 2> 120136 1518 0 NONE  0
802.3 RX  99995979400000 1214400000 DATA  <1 2> <1 2> 120136 1518 0 NONE  0
802.3 TX  99997202400000 1214400000 DATA  <1 2> <1 2> 120137 1518 0 NONE  0
802.3 RX  99997203400000 1214400000 DATA  <1 2> <1 2> 120137 1518 0 NONE  0
802.3 TX  99998417800000 56000000 DATA  <2 1> <2 1> 120193 70 0 NONE  0
802.3 RX  99998418800000 56000000 DATA  <2 1> <2 1> 120193 70 0 NONE  0
802.3 TX  99998426400000 1214400000 DATA  <1 2> <1 2> 120138 1518 0 NONE  0
802.3 RX  99998427400000 1214400000 DATA  <1 2> <1 2> 120138 1518 0 NONE  0
[root@localhost Chapter5]#
```

第六章：TIMER

本章重點：
(1)、TIMER 練習
(2)、TIMER 用法說明
(3)、GETCURRENTTIME、GETNODETIME、SEC_TO_TICK、MILLI_TO_TICK、BASE_OBJTYPE、POINTER_TO_MEMBER 等 API 的說明

下載練習：


Chapter6.tar.bz2

Timer 在模擬中是一個重要物件，尤其是模擬時，需要控制一些特定函數何時被觸發、啟動，可以說是一個重要物件，這個物件中，提供了像 **init()**、**start()**、**cancel()**、**expire()**等函數來控制，以下用一個簡單的範例來說明：

■ 練習 6-1

這個範例中，一樣用 user_module01 的拓撲檔(user_module01.xtpl)。

接著在 user_module01.h 檔是宣告了一個 timer 物件為 myTimer，timerObj 這個是模擬器已經定義好的物件，只要 include timer.h 即可使用。
在 public 區塊內宣告一個 timeout 的函式，這個函式是希望在 timer 被觸發時，可以被啟動執行的函式。

user_module01.h 檔修改部分如下紅字所示：

```
#ifndef __user_module_01_h__
#define __user_module_01_h__

#include <event.h>
#include <object.h>
#include <timer.h>

class UserModule01 : public NslObject {
```

```
private:
        int             Number;
        char            *String;
        timerObj        myTimer;

public:

        UserModule01(u_int32_t type, u_int32_t id, struct plist* pl, const char *name);
        ~UserModule01();

        int             init();
        int             command(int argc, const char *argv[]);
        int             recv(ePacket_ *pkt);
        int             send(ePacket_ *pkt);
        void            timeout();
};

#endif    /* __user_module_01_h__ */
```

接著在 user_module01.cc 檔中，定義這個 timeout 的函式被呼叫到時，可以列印一些資訊如目前的虛擬時間，以及每個 node 上的虛擬時間。**GetCurrentTime** 是用來取得模擬器所使用的虛擬時間，單位是 1 個 tick 為 1 picoosecond(10^{-12})。而在模擬器中，每個 node 上還有自己的時間，使用 **GetNodeCurrentTime**()這個 API 來取得。

另外在 init()函式，要加入 myTimer 物件的設定。
首先先宣告兩個變數，first_timeout_in_tick 是第一次執行的時間點；另一個 subsequent_timeout_in_tick 是第一次執行後每次 timeout 的時間間隔。

user_module01.cc 檔中修改部分如下紅字所示：

```
#include <stdlib.h>
#include <estinet_api.h>
#include <module/user-defined/user_module_01.h>

MODULE_GENERATOR(UserModule01);

UserModule01::UserModule01(u_int32_t type, u_int32_t id, struct plist* pl, const char *name)
        : NslObject(type, id, pl, name) {
```

```
        vBind_int("myNumber", &Number);
        vBind_char_str("myString", &String);
        REG_VAR("shared-number", &Number);
}

UserModule01::~UserModule01(){}

int UserModule01::init() {
        u_int64_t   first_timeout_in_tick;
        u_int64_t   subsequent_timeout_in_tick;
        BASE_OBJTYPE(mem_func);

        SEC_TO_TICK(first_timeout_in_tick, 3);
        MILLI_TO_TICK(subsequent_timeout_in_tick, 500);

        mem_func = POINTER_TO_MEMBER(UserModule01, timeout);
        myTimer.setCallOutObj(this, mem_func);
        myTimer.start(first_timeout_in_tick, subsequent_timeout_in_tick);

        return(NslObject::init());
}

int UserModule01::command(int argc, const char *argv[]) {
        return(NslObject::command(argc, argv));
}

int UserModule01::recv(ePacket_ *pkt) {
        return(NslObject::recv(pkt));
}

int UserModule01::send(ePacket_ *pkt) {
        return(NslObject::send(pkt));
}

void UserModule01::timeout() {

        printf("\e[1;33;40mExercise 6-1: Timeout (%llu) (%llu)\e[m\n",
                        GetCurrentTime(), GetNodeCurrentTime(get_nid()));
```

```
        return;
}
```

BASE_OBJTYPE(mem_func);
這行則是宣告一個基本物件的指標，等一下用來指向某一個模組的某個函式。

SEC_TO_TICK(first_timeout_in_tick, 3);
這個 SEC_TO_TICK 是可以把秒數轉為 tick 的單位，例如此例中的 3 秒，轉為 3000000000000 tick，帶入到 first_timeout_in_tick 這個變數中。

MILLI_TO_TICK(subsequent_timeout_in_tick, 500);
這個 MILLI_TO_TICK 的 API 與 SEC_TO_TICK 類似，但是是將 MILLISECOND 的單位，轉為 tick 單位，例如此例中的 500 millisecond (即 0.5 秒)，轉為 500000000000tick，將此值帶入到 subsequent_timeout_in_tick 這個變數中。

mem_func = POINTER_TO_MEMBER(UserModule01, timeout);
接著讓 mem_func 這個基本物件使用 POINTER_TO_MEMBER 這個 API，來指向 UserModule01 模組中的 timeout 的位置。

myTimer.setCallOutObj(this, mem_func);
接著設定 myTimer 時間到時，要呼叫這個 mem_func 所指到的函式。

myTimer.start(first_timeout_in_tick, subsequent_timeout_in_tick);
最後設定 myTimer 啟動的時間以及執行後再重覆執行的時間循環。第一個參數是第一次啟動的時間點；第二個參數是重覆執行的時間單位。

因此可以知道，這個 timer 在 3 秒會被啟動，接著每 0.5 秒重覆被執行，如下圖所示。

執行結果如下：

```
net.ipv6.conf.eth0.disable_ipv6 = 0
net.ipv6.conf.eth0.autoconf = 0
Current Time:    0.00 sec   Event#: <Insert:23, Dequeue:5, Rest:18>
current ticks= 100100000, run "1 sh init_daemon.sh"
current ticks= 100200000, run "2 sh init_daemon.sh"
Current Time:    1.00 sec   Event#: <Insert:1048, Dequeue:1048, Rest:18>
Current Time:    2.00 sec   Event#: <Insert:1053, Dequeue:1053, Rest:18>
Current Time:    3.00 sec   Event#: <Insert:1041, Dequeue:1041, Rest:18>
Exercise 6-1: Timeout (3000000000000) (3000000004383)
Exercise 6-1: Timeout (3000000000000) (3000000003677)
Exercise 6-1: Timeout (3500000000000) (3500000004383)
Exercise 6-1: Timeout (3500000000000) (3500000003677)
Current Time:    4.00 sec   Event#: <Insert:1043, Dequeue:1043, Rest:18>
Exercise 6-1: Timeout (4000000000000) (4000000004383)
Exercise 6-1: Timeout (4000000000000) (4000000003677)
Exercise 6-1: Timeout (4500000000000) (4500000004383)
Exercise 6-1: Timeout (4500000000000) (4500000003677)
Current Time:    5.00 sec   Event#: <Insert:1041, Dequeue:1041, Rest:18>
Exercise 6-1: Timeout (5000000000000) (5000000004383)
Exercise 6-1: Timeout (5000000000000) (5000000003677)
Exercise 6-1: Timeout (5500000000000) (5500000004383)
Exercise 6-1: Timeout (5500000000000) (5500000003677)
Current Time:    6.00 sec   Event#: <Insert:1045, Dequeue:1045, Rest:18>
Exercise 6-1: Timeout (6000000000000) (6000000004383)
Exercise 6-1: Timeout (6000000000000) (6000000003677)
Exercise 6-1: Timeout (6500000000000) (6500000004383)
Exercise 6-1: Timeout (6500000000000) (6500000003677)
Current Time:    7.00 sec   Event#: <Insert:1041, Dequeue:1041, Rest:18>
Exercise 6-1: Timeout (7000000000000) (7000000004383)
Exercise 6-1: Timeout (7000000000000) (7000000003677)
Exercise 6-1: Timeout (7500000000000) (7500000004383)
```

■ 練習 6-2：延伸練習 timer

請使用者練習加入一個 timer 呼叫函式 timeout，第 5 秒被啟動，之後每隔 2 秒執行，參考答案如下所示：

user_module01.cc 檔中修改部分如下紅字所示：

```
#include <stdlib.h>
#include <estinet_api.h>
#include <module/user-defined/user_module_01.h>

MODULE_GENERATOR(UserModule01);

UserModule01::UserModule01(u_int32_t type, u_int32_t id, struct iflist* ifl, const char
*name): NslObject(type, id, ifl, name) {
        vBind_int("myNumber", &Number);
        vBind_char_str("myString", &String);
        REG_VAR("shared-number", &Number);
```

```
}

UserModule01::~UserModule01(){}

int UserModule01::init() {
        u_int64_t    first_timeout_in_tick;
        u_int64_t    subsequent_timeout_in_tick;
        BASE_OBJTYPE(mem_func);

        SEC_TO_TICK(first_timeout_in_tick, 5);
        SEC_TO_TICK(subsequent_timeout_in_tick, 2);

        mem_func = POINTER_TO_MEMBER(UserModule01, timeout);
        myTimer.setCallOutObj(this, mem_func);
        myTimer.start(first_timeout_in_tick, subsequent_timeout_in_tick);

        return(NslObject::init());
}

int UserModule01::command(int argc, const char *argv[]) {
        return(NslObject::command(argc, argv));
}

int UserModule01::recv(ePacket_ *pkt) {
        return(NslObject::recv(pkt));
}

int UserModule01::send(ePacket_ *pkt) {
        return(NslObject::send(pkt));
}

void UserModule01::timeout() {

        printf("\e[1;33;40mExercise 6-2: Timeout (%llu) (%llu)\e[m\n",
                        GetCurrentTime(), GetNodeCurrentTime(get_nid()));
        return;
}
```

```
current ticks= 100200000, run "2 sh init_daemon.sh"
Current Time:     1.00 sec    Event#: <Insert:1048, Dequeue:1048, Rest:18>
Current Time:     2.00 sec    Event#: <Insert:1052, Dequeue:1052, Rest:18>
Current Time:     3.00 sec    Event#: <Insert:1042, Dequeue:1042, Rest:18>
Current Time:     4.00 sec    Event#: <Insert:1043, Dequeue:1043, Rest:18>
Current Time:     5.00 sec    Event#: <Insert:1041, Dequeue:1041, Rest:18>
Exercise 6-2: Timeout (5000000000000) (5000000004383)
Exercise 6-2: Timeout (5000000000000) (5000000003677)
Current Time:     6.00 sec    Event#: <Insert:1045, Dequeue:1045, Rest:18>
Current Time:     7.00 sec    Event#: <Insert:1041, Dequeue:1041, Rest:18>
Exercise 6-2: Timeout (7000000000000) (7000000004383)
Exercise 6-2: Timeout (7000000000000) (7000000003677)
Current Time:     8.00 sec    Event#: <Insert:1043, Dequeue:1043, Rest:18>
Current Time:     9.00 sec    Event#: <Insert:1041, Dequeue:1041, Rest:18>
Exercise 6-2: Timeout (9000000000000) (9000000004383)
Exercise 6-2: Timeout (9000000000000) (9000000003677)
Current Time:    10.00 sec    Event#: <Insert:1045, Dequeue:1045, Rest:18>
Current Time:    11.00 sec    Event#: <Insert:1041, Dequeue:1041, Rest:18>
Exercise 6-2: Timeout (11000000000000) (11000000004383)
Exercise 6-2: Timeout (11000000000000) (11000000003677)
Current Time:    12.00 sec    Event#: <Insert:1043, Dequeue:1043, Rest:18>
Current Time:    13.00 sec    Event#: <Insert:1041, Dequeue:1041, Rest:18>
Exercise 6-2: Timeout (13000000000000) (13000000004383)
Exercise 6-2: Timeout (13000000000000) (13000000003677)
Current Time:    14.00 sec    Event#: <Insert:1043, Dequeue:1043, Rest:18>
Current Time:    15.00 sec    Event#: <Insert:1041, Dequeue:1041, Rest:18>
Exercise 6-2: Timeout (15000000000000) (15000000004383)
Exercise 6-2: Timeout (15000000000000) (15000000003677)
```

■ 練習 6-3：延伸練習 timer

接著練習在 mytimer 呼叫函式 timeout，第 3 秒被啟動，之後每隔 0.5 秒執行，第 5 秒暫停，第 7 秒恢復，第 9 秒被取消
這需要用到 timer 物件中的 **cancel()**、**pause()**、以及 **resume()**

首先還需要在 user_module01.h 檔中多宣告一個 timer 跟一個函式，讓 timer 被暫停後，由另一個 timer 幫忙恢復。如下面藍字所示。

```
#ifndef __user_module_01_h__
#define __user_module_01_h__

#include <event.h>
#include <object.h>
#include <timer.h>

class UserModule01 : public NslObject {

 private:
```

```
        int             Number;
        char            *String;
        timerObj        myTimer, resumeTimer;

    public:

        UserModule01(u_int32_t type, u_int32_t id, struct plist* pl, const char *name);
        ~UserModule01();

        int         init();
        int         command(int argc, const char *argv[]);
        int         recv(ePacket_ *pkt);
        int         send(ePacket_ *pkt);
        void        timeout();
        void        resumetimer();
};

#endif    /* __user_module_01_h__ */
```

接著修改 user_module01.cc，如下紅字所示：

```
#include <stdlib.h>
#include <estinet_api.h>
#include <module/user-defined/user_module_01.h>

MODULE_GENERATOR(UserModule01);

UserModule01::UserModule01(u_int32_t type, u_int32_t id, struct plist* pl, const char
*name)
    : NslObject(type, id, pl, name) {}

UserModule01::~UserModule01(){}



int UserModule01::init() {
    u_int64_t   first_timeout_in_tick;
    u_int64_t   subsequent_timeout_in_tick;
    BASE_OBJTYPE(mem_func);
```

```
        SEC_TO_TICK(first_timeout_in_tick, 3);
        MILLI_TO_TICK(subsequent_timeout_in_tick, 500);

        mem_func = POINTER_TO_MEMBER(UserModule01, timeout);
        myTimer.setCallOutObj(this, mem_func);
        myTimer.start(first_timeout_in_tick, subsequent_timeout_in_tick);

        u_int64_t   resume_timeout_in_tick;
        BASE_OBJTYPE(mem_func2);
        mem_func2 = POINTER_TO_MEMBER(UserModule01, resumetimer);
        SEC_TO_TICK(resume_timeout_in_tick, 7);
        resumeTimer.setCallOutObj(this, mem_func2);
        resumeTimer.start(resume_timeout_in_tick, 0);

        return(NslObject::init());
}

int UserModule01::command(int argc, const char *argv[]) {
        return(NslObject::command(argc, argv));
}

int UserModule01::recv(ePacket_ *pkt) {
        return(NslObject::recv(pkt));
}

int UserModule01::send(ePacket_ *pkt) {
        return(NslObject::send(pkt));
}

void UserModule01::timeout() {

        printf("\e[1;33;40mExercise 6-3: Timeout (%llu) (%llu) Get time timer expired time:
%llu \e[m\n", GetCurrentTime(), GetNodeCurrentTime(get_nid()), myTimer.expire());

        if(GetCurrentTime() == 9000000000000)
        {
                myTimer.cancel();
```

```
        }
        else if(GetCurrentTime() == 5000000000000)
        {
                myTimer.pause();
        }

        return;
}

void UserModule01::resumetimer() {

        myTimer.resume();
        return;
}
```

修改 timeout()函式，在得到模擬時間為 5 秒時，將 mytimer 暫停，而在模擬時間 9 秒時，將 mytimer 取消。

列印中加入 myTimer()的 expire()函式，可以看到下次 myTimer 被啟動的時間。

接著定義 resumetimer()函式則是用來恢復 mytimer

在 init 下多加上藍色的區段，定義在第 7 秒由 resumeTimer 啟動 resumetimer()函式來恢復 myTimer()

值得注意的是 resumeTimer 後面的 interval 是帶 0，表示這個 Timer 只執行 1 次即可。

可以從下面執行結果中，看到第 5 秒時列印完時間後，接著就被暫停了，等到第 7 秒時，由 resumeTimer 恢復 myTimer 後，又開始呼叫 timeout 函式列印，直到第 9 秒 timeout 函式時，列印後直接取消 myTimer。

```
current ticks= 100200000, run "2 sh init_daemon.sh"
Current Time:    1.00 sec   Event#: <Insert:1048, Dequeue:1048, Rest:18>
Current Time:    2.00 sec   Event#: <Insert:1052, Dequeue:1052, Rest:18>
Current Time:    3.00 sec   Event#: <Insert:1042, Dequeue:1042, Rest:18>
Exercise 6-3: Timeout (3000000000000) (3000000004383) Get time timer expired time: 3500000000000
Exercise 6-3: Timeout (3000000000000) (3000000003677) Get time timer expired time: 3500000000000
Exercise 6-3: Timeout (3500000000000) (3500000004383) Get time timer expired time: 4000000000000
Exercise 6-3: Timeout (3500000000000) (3500000003677) Get time timer expired time: 4000000000000
Current Time:    4.00 sec   Event#: <Insert:1043, Dequeue:1043, Rest:18>
Exercise 6-3: Timeout (4000000000000) (4000000004383) Get time timer expired time: 4500000000000
Exercise 6-3: Timeout (4000000000000) (4000000003677) Get time timer expired time: 4500000000000
Exercise 6-3: Timeout (4500000000000) (4500000004383) Get time timer expired time: 5000000000000
Exercise 6-3: Timeout (4500000000000) (4500000003677) Get time timer expired time: 5000000000000
Current Time:    5.00 sec   Event#: <Insert:1041, Dequeue:1041, Rest:18>
Exercise 6-3: Timeout (5000000000000) (5000000004383) Get time timer expired time: 5500000000000
Exercise 6-3: Timeout (5000000000000) (5000000003677) Get time timer expired time: 5500000000000
Current Time:    6.00 sec   Event#: <Insert:1045, Dequeue:1045, Rest:18>
Current Time:    7.00 sec   Event#: <Insert:1041, Dequeue:1041, Rest:18>
Exercise 6-3: Timeout (7500000000000) (7500000004383) Get time timer expired time: 8000000000000
Exercise 6-3: Timeout (7500000000000) (7500000003677) Get time timer expired time: 8000000000000
Current Time:    8.00 sec   Event#: <Insert:1043, Dequeue:1043, Rest:18>
Exercise 6-3: Timeout (8000000000000) (8000000004383) Get time timer expired time: 8500000000000
Exercise 6-3: Timeout (8000000000000) (8000000003677) Get time timer expired time: 8500000000000
Exercise 6-3: Timeout (8500000000000) (8500000004383) Get time timer expired time: 9000000000000
Exercise 6-3: Timeout (8500000000000) (8500000003677) Get time timer expired time: 9000000000000
Current Time:    9.00 sec   Event#: <Insert:1041, Dequeue:1041, Rest:18>
Exercise 6-3: Timeout (9000000000000) (9000000004383) Get time timer expired time: 9500000000000
Exercise 6-3: Timeout (9000000000000) (9000000003677) Get time timer expired time: 9500000000000
Current Time:   10.00 sec   Event#: <Insert:1045, Dequeue:1045, Rest:18>
Current Time:   11.00 sec   Event#: <Insert:1041, Dequeue:1041, Rest:18>
Current Time:   12.00 sec   Event#: <Insert:1043, Dequeue:1043, Rest:18>
Current Time:   13.00 sec   Event#: <Insert:1041, Dequeue:1041, Rest:18>
Current Time:   14.00 sec   Event#: <Insert:1043, Dequeue:1043, Rest:18>
Current Time:   15.00 sec   Event#: <Insert:1041, Dequeue:1041, Rest:18>
Current Time:   16.00 sec   Event#: <Insert:1043, Dequeue:1043, Rest:18>
Current Time:   17.00 sec   Event#: <Insert:1041, Dequeue:1041, Rest:18>
Current Time:   18.00 sec   Event#: <Insert:1043, Dequeue:1043, Rest:18>
Current Time:   19.00 sec   Event#: <Insert:1041, Dequeue:1041, Rest:18>
```

第七章：EVENT

本章重點：
(1)、EVENT 練習
(2)、EVENT 用法說明

下載練習：


Chapter7.tar.bz2

模擬器中，也可以使用 EVENT 的資料結構來觸發事件，在模擬器中其實像 PACKET 其實都是用 EVENT 去包裝起來的。每個封包都是一個 EVENT。
而 timer 其實也是 event 結構的延伸，timer 的本質上也是一個 event，但多了許多為 timer 設計使用的函式。

本章要學會怎麼自己創造一個 EVENT，來觸發一個函式執行，之後再將此 EVENT 空間釋放

下面用一個例子說明 event 的用法，我們創造一個 event 並宣告一個 print_event()的函式，設定在第一秒時讓模擬器中的 event 去觸發此函式來列印。

■ 練習 7-1
同樣使用第一章的範例拓撲(user_module01.xtpl)。

修改 user_module01.h，如下紅字所示：

```
#ifndef __user_module_01_h__
#define __user_module_01_h__

#include <event.h>
#include <object.h>

class UserModule01 : public NslObject {

  private:
```

```
        int             Number;
        char            *String;

 public:

        UserModule01(u_int32_t type, u_int32_t id, struct plist* pl, const char *name);
        ~UserModule01();

        int         init();
        int         command(int argc, const char *argv[]);
        int         recv(ePacket_ *pkt);
        int         send(ePacket_ *pkt);
        void        print_event(Event_ *ep);
};

#endif    /* __user_module_01_h__ */
```

修改 user_module01.cc，如下紅字所示：

```
#include <stdlib.h>
#include <estinet_api.h>
#include <module/user-defined/user_module_01.h>

MODULE_GENERATOR(UserModule01);

UserModule01::UserModule01(u_int32_t type, u_int32_t id, struct plist* pl, const char *name)
        : NslObject(type, id, pl, name) {
        vBind_int("myNumber", &Number);
        vBind_char_str("myString", &String);
        REG_VAR("shared-number", &Number);
}

UserModule01::~UserModule01(){}

int UserModule01::init() {

        Event_ *ep;
        ep = createEvent();
```

```
        u_int64_t expire;
        BASE_OBJTYPE(mem_func3);

        SEC_TO_TICK(expire, 1);
        mem_func3 = POINTER_TO_MEMBER(UserModule01, print_event);

        setObjEvent(ep,
                        expire,
                        0,
                        this,
                        mem_func3,
                        (void *)0
                    );

        return(NslObject::init());

}

int UserModule01::command(int argc, const char *argv[]) {
        return(NslObject::command(argc, argv));
}

int UserModule01::recv(ePacket_ *pkt) {
        return(NslObject::recv(pkt));
}

int UserModule01::send(ePacket_ *pkt) {
        return(NslObject::send(pkt));
}

void UserModule01::print_event(Event_ *ep) {
        printf("\e[1;33;40mExercise 7-1: Timeout (%llu) (%llu)\e[m\n",
                        GetCurrentTime(), GetNodeCurrentTime(get_nid()));
        freeEvent(ep);
        return;
}
```

createEvent()用來創造一個新的 event。

而 freeEvent()函式則是用來釋放這個 event 的空間。

setObjEvent 則是設定此 event 的細節，第一個參數是 event 的指標，用來指向某個 event，第二個參數是執行的時間，第三個參數是 interval，如果是 0 表示只執行一次，第四個參數是指執行的物件，而第五個函式是指定處理此 event 的函式或物件，第六個參數是一些額外的 data，像封包的資料就會帶入於此處

下圖就是執行結果，我們可以看到，在第一秒時讓模擬器中的 event 去觸發此函式來列印。因為第三個參數帶入 0，因此只執行一次。

```
current ticks= 100100000, run "1 sh init_daemon.sh"
current ticks= 100200000, run "2 sh init_daemon.sh"
Exercise 7-1: Timeout (1000000000000) (1000000004383)
Current Time:    1.00 sec   Event#: <Insert:1056, Dequeue:1055, Rest:19>
Exercise 7-1: Timeout (1000000000000) (1000000003677)
Current Time:    2.00 sec   Event#: <Insert:1046, Dequeue:1047, Rest:18>
Current Time:    3.00 sec   Event#: <Insert:1043, Dequeue:1043, Rest:18>
Current Time:    4.00 sec   Event#: <Insert:1042, Dequeue:1042, Rest:18>
Current Time:    5.00 sec   Event#: <Insert:1042, Dequeue:1042, Rest:18>
Current Time:    6.00 sec   Event#: <Insert:1044, Dequeue:1044, Rest:18>
Current Time:    7.00 sec   Event#: <Insert:1042, Dequeue:1042, Rest:18>
Current Time:    8.00 sec   Event#: <Insert:1042, Dequeue:1042, Rest:18>
Current Time:    9.00 sec   Event#: <Insert:1042, Dequeue:1042, Rest:18>
Current Time:   10.00 sec   Event#: <Insert:1044, Dequeue:1044, Rest:18>
Current Time:   11.00 sec   Event#: <Insert:1042, Dequeue:1042, Rest:18>
Current Time:   12.00 sec   Event#: <Insert:1042, Dequeue:1042, Rest:18>
Current Time:   13.00 sec   Event#: <Insert:1042, Dequeue:1042, Rest:18>
Current Time:   14.00 sec   Event#: <Insert:1042, Dequeue:1042, Rest:18>
Current Time:   15.00 sec   Event#: <Insert:1042, Dequeue:1042, Rest:18>
Current Time:   16.00 sec   Event#: <Insert:1042, Dequeue:1042, Rest:18>
Current Time:   17.00 sec   Event#: <Insert:1042, Dequeue:1042, Rest:18>
```

- 練習 7-2：
  若希望這個 EVENT，週期性的執行，可以使用下面修改 user_module01.cc 後的程式碼，如紅字所示：

```
#include <stdlib.h>
#include <estinet_api.h>
#include <module/user-defined/user_module_01.h>

MODULE_GENERATOR(UserModule01);

UserModule01::UserModule01(u_int32_t type, u_int32_t id, struct iflist* ifl, const char
*name): NslObject(type, id, ifl, name) {
```

```
        vBind_int("myNumber", &Number);
        vBind_char_str("myString", &String);
        REG_VAR("shared-number", &Number);
}

UserModule01::~UserModule01(){}

int UserModule01::init() {

        Event_ *ep;
        ep = createEvent();

        u_int64_t expire;
        BASE_OBJTYPE(mem_func3);

        SEC_TO_TICK(expire, 1);
        mem_func3 = POINTER_TO_MEMBER(UserModule01, print_event);

        setObjEvent(ep,
                        expire,
                        expire,
                        this,
                        mem_func3,
                        (void *)0
                    );

        return(NslObject::init());

}

int UserModule01::command(int argc, const char *argv[]) {
        return(NslObject::command(argc, argv));
}

int UserModule01::recv(ePacket_ *pkt) {
        return(NslObject::recv(pkt));
}
```

```
int UserModule01::send(ePacket_ *pkt) {
        return(NslObject::send(pkt));
}

void UserModule01::print_event(Event_ *ep) {
        printf("\e[1;33;40mExercise 7-2: Timeout (%llu) (%llu)\e[m\n",
                        GetCurrentTime(), GetNodeCurrentTime(get_nid()));
        //freeEvent(ep);
        setEventReuse(ep);
        return;
}
```

執行結果如下：

```
current ticks= 100200000, run "2 sh init_daemon.sh"
Exercise 7-2: Timeout (1000000000000) (1000000004383)
Current Time:     1.00 sec    Event#: <Insert:1057, Dequeue:1055, Rest:20>
Exercise 7-2: Timeout (1000000000000) (1000000003677)
Exercise 7-2: Timeout (2000000000000) (2000000003677)
Exercise 7-2: Timeout (2000000000000) (2000000004383)
Current Time:     2.00 sec    Event#: <Insert:1054, Dequeue:1054, Rest:20>
Exercise 7-2: Timeout (3000000000000) (3000000003677)
Exercise 7-2: Timeout (3000000000000) (3000000004383)
Current Time:     3.00 sec    Event#: <Insert:1046, Dequeue:1046, Rest:20>
Current Time:     4.00 sec    Event#: <Insert:1037, Dequeue:1037, Rest:20>
Exercise 7-2: Timeout (4000000000000) (4000000003677)
Exercise 7-2: Timeout (4000000000000) (4000000004383)
Exercise 7-2: Timeout (5000000000000) (5000000004383)
Current Time:     5.00 sec    Event#: <Insert:1049, Dequeue:1049, Rest:20>
Exercise 7-2: Timeout (5000000000000) (5000000003677)
Exercise 7-2: Timeout (6000000000000) (6000000003677)
Exercise 7-2: Timeout (6000000000000) (6000000004383)
Current Time:     6.00 sec    Event#: <Insert:1047, Dequeue:1047, Rest:20>
Exercise 7-2: Timeout (7000000000000) (7000000003677)
Exercise 7-2: Timeout (7000000000000) (7000000004383)
Current Time:     7.00 sec    Event#: <Insert:1045, Dequeue:1045, Rest:20>
Current Time:     8.00 sec    Event#: <Insert:1037, Dequeue:1037, Rest:20>
Exercise 7-2: Timeout (8000000000000) (8000000003677)
Exercise 7-2: Timeout (8000000000000) (8000000004383)
Exercise 7-2: Timeout (9000000000000) (9000000004383)
Current Time:     9.00 sec    Event#: <Insert:1049, Dequeue:1049, Rest:20>
Exercise 7-2: Timeout (9000000000000) (9000000003677)
```

除了要加入第三個參數外，另外還要搭配 setEventReuse 這個 API。因此若需要週期性的執行，不妨選擇使用 timer 物件較為理想。

跟封包有關的 event 結構，在下一個章節進行介紹。

第八章：PACKET

本章重點：
(1)、PACKET 練習
(2)、PACKET 用法說明

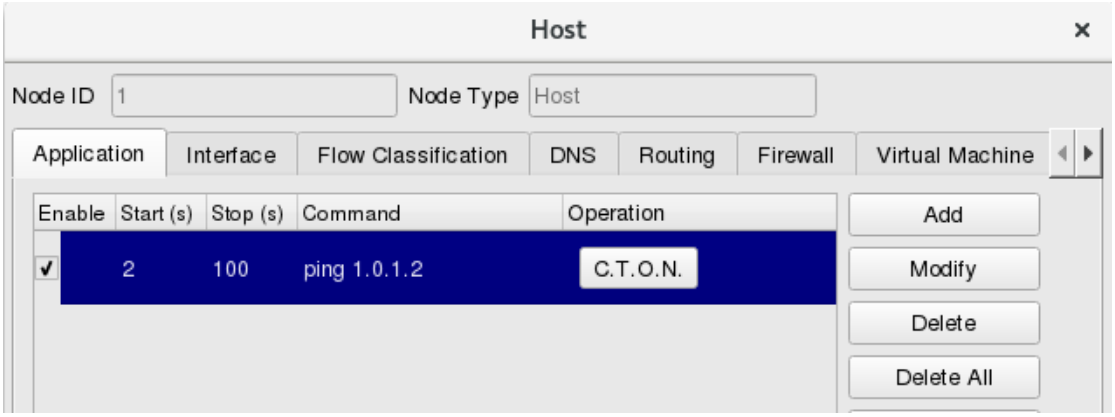
下載練習：

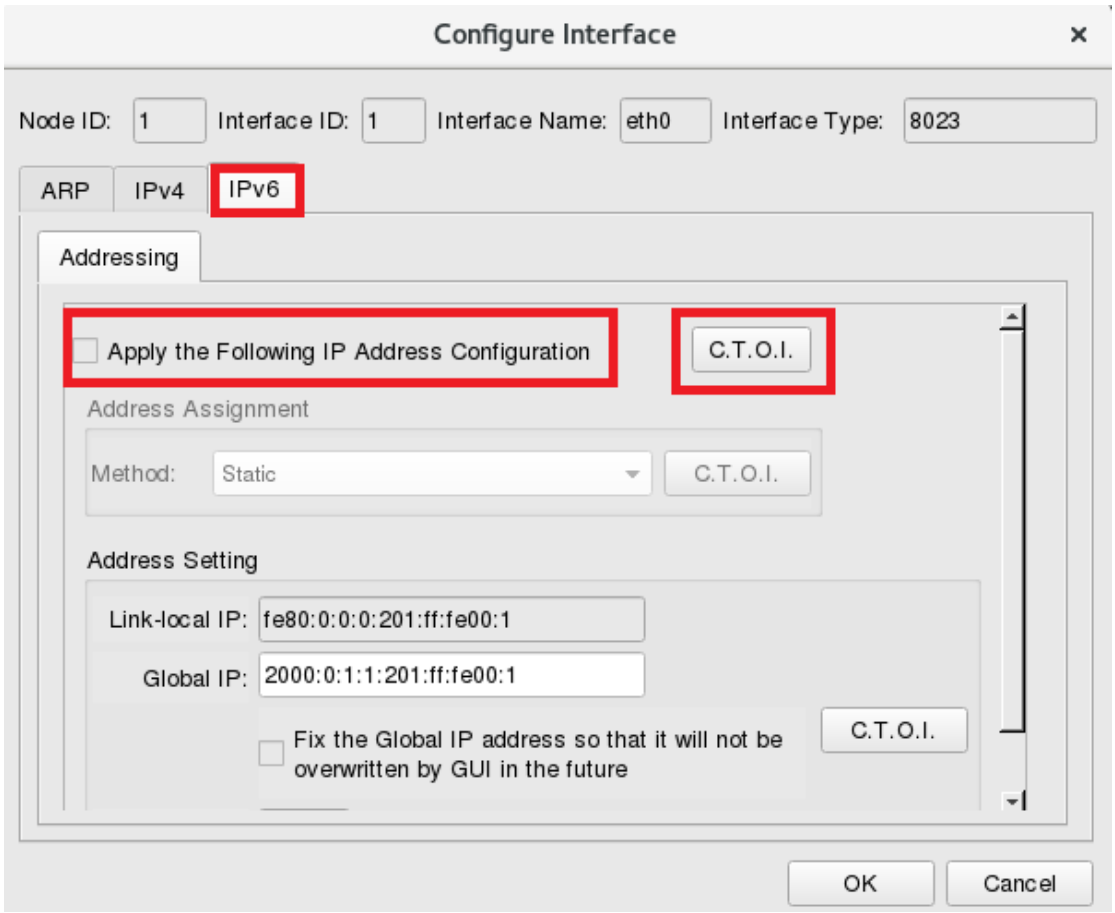
Chapter8.tar.bz2

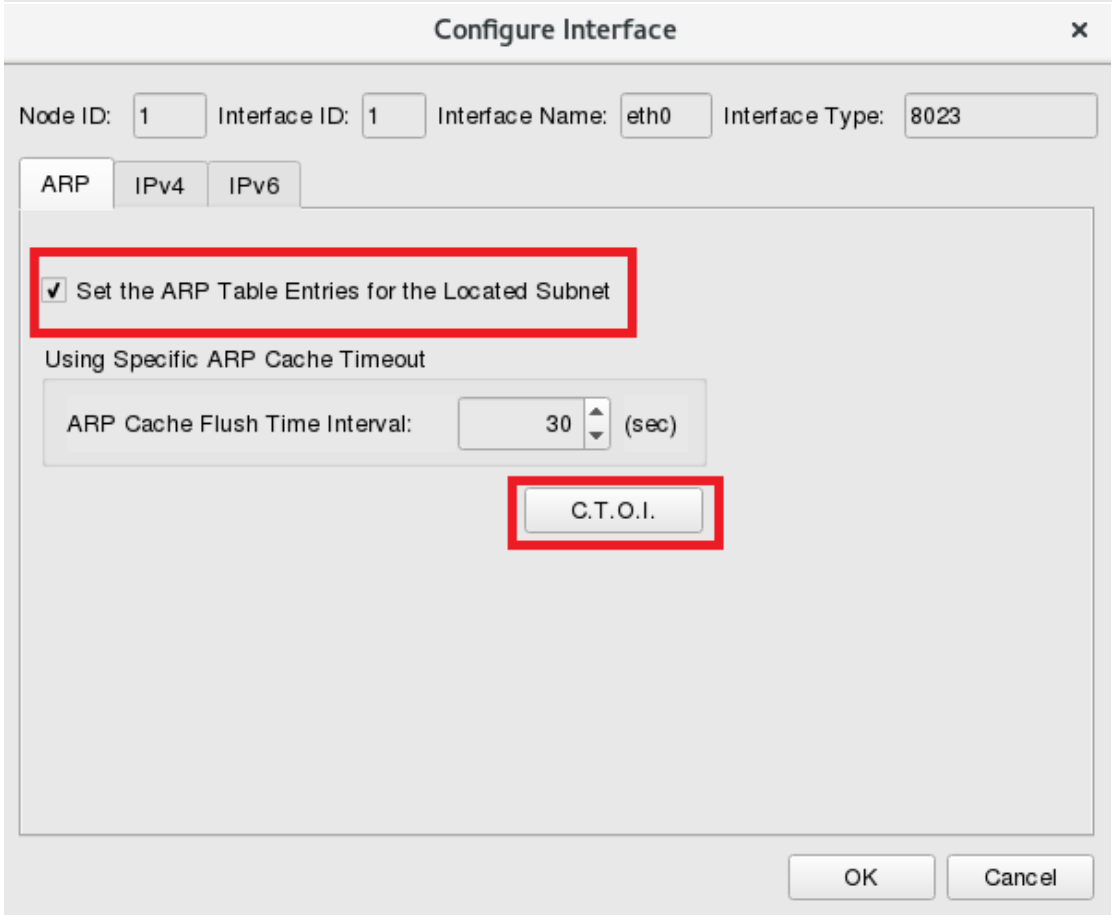
在網路模擬中，最重要的是封包的處理，在模擬器中，主要是有一個處理的物件，一開始會由 Interface 模組，將封包包裝在 Packet 結構中，再循序經由每一個模組中的 send()進行處理(模組接收與傳送的架構，可以查看第四章中說明)。實際上 PACKET 物件使用 EVENT 的結構，因此也具有 EVENT 的特性。以下例來使用及說明這個 packet 物件結構。

■ 練習 8-1：

同樣使用第一章的範例拓撲(user_module01.xtpl)，並加入通訊流如下：
首先使用 Host1 ping Host2，在 Host1 中加入 ping 1.0.1.2。在 Host1 與 Host2 上都關閉 ARP 與 IPv6 介面。




Configure Interface
Node ID: 1
Interface ID: 1
Interface Name: eth0
Interface Type: 8023
ARP
IPv4
IPv6
Addressing
Apply the Following IP Address Configuration
C.T.O.I.
Address Assignment
Method: Static
C.T.O.I.
Address Setting
Link-local IP: fe80:0:0:0:201:ff:fe00:1
Global IP: 2000:0:1:1:201:ff:fe00:1
Fix the Global IP address so that it will not be overwritten by GUI in the future
C.T.O.I.
OK
Cancel


Configure Interface
Node ID: 1
Interface ID: 1
Interface Name: eth0
Interface Type: 8023
ARP
IPv4
IPv6
Set the ARP Table Entries for the Located Subnet
Using Specific ARP Cache Timeout
ARP Cache Flush Time Interval: 30 (sec)
C.T.O.I.
OK
Cancel

接著在 user_module01.h 檔中宣告一個函式 pkt_delay，如下紅字所示：

```
#ifndef __user_module_01_h__
#define __user_module_01_h__

#include <event.h>
#include <object.h>

class UserModule01 : public NslObject {

 private:
        int             Number;
        char            *String;

 public:

        UserModule01(u_int32_t type, u_int32_t id, struct plist* pl, const char *name);
        ~UserModule01();

        int             init();
        int             command(int argc, const char *argv[]);
        int             recv(ePacket_ *pkt);
        int             send(ePacket_ *pkt);
        void            pkt_delay(ePacket_ *pkt);
};

#endif    /* __user_module_01_h__ */
```

接著修改 user_module01.cc，修改部分如下紅字所示：

```
#include <stdlib.h>
#include <estinet_api.h>
#include <module/user-defined/user_module_01.h>
#include <packet.h>
#include <ethernet.h>
#include <ip.h>

MODULE_GENERATOR(UserModule01);
```

```
UserModule01::UserModule01(u_int32_t type, u_int32_t id, struct plist* pl, const char *name)
        : NslObject(type, id, pl, name) {
        vBind_int("myNumber", &Number);
        vBind_char_str("myString", &String);
        REG_VAR("shared-number", &Number);
}

UserModule01::~UserModule01(){}

int UserModule01::init() {
        return(NslObject::init());
}

int UserModule01::command(int argc, const char *argv[]) {
        return(NslObject::command(argc, argv));
}

int UserModule01::recv(ePacket_ *pkt) {
        return(NslObject::recv(pkt));
}

int UserModule01::send(ePacket_ *pkt) {
        u_int64_t    current_time_in_tick;
        u_int64_t    delay_time_in_tick;
        BASE_OBJTYPE(mem_func);

        current_time_in_tick = GetCurrentTime();
        SEC_TO_TICK(delay_time_in_tick, 1);
        mem_func = POINTER_TO_MEMBER(UserModule01, pkt_delay);
        setObjEvent(pkt, current_time_in_tick + delay_time_in_tick,
                0, this, mem_func, (void *)pkt->DataInfo_);
        return(1);

}

void UserModule01::pkt_delay(ePacket_ *pkt) {

        Packet                  *packet;
```

```
        struct ether_header     *eh;
        char                    src_mac_str[18];
        char                    dst_mac_str[18];

        struct ip               *iph;
        char                    src_ip_str[16];
        char                    dst_ip_str[16];

        if(pkt != NULL && pkt->DataInfo_ != NULL) {
                packet = (Packet *)pkt->DataInfo_;
                eh = (struct ether_header *)packet->pkt_get();

                macaddr_to_str(eh->ether_shost, src_mac_str);
                macaddr_to_str(eh->ether_dhost, dst_mac_str);

                printf("\e[1;36;40mExercise 8-1: Src Mac: %s, Dst Mac: %s\e[m\n",
                        src_mac_str, dst_mac_str);

                iph = (struct ip *)packet->pkt_sget();
                if(iph != NULL) {
                        ipv4addr_to_str(iph->ip_src, src_ip_str);
                        ipv4addr_to_str(iph->ip_dst, dst_ip_str);
                        printf("\e[1;36;40mExercise 8-1: Src IP: %s, Dst IP: %s\e[m\n",
                        src_ip_str, dst_ip_str);
                }
        }

        NslObject::send(pkt);
}
```

在 send()函式中，接收到通訊流的封包後，先不要送出去，而將這個封包中的 event 設定在 1 秒後，觸發了 pkt_delay 這個函式進行處理，才送給下一個模組。

這樣可以做到封包延遲的效果。也可以發現其實在模組中的封包是由 event 物件所包裝。

另外，封包結構中的 DataInfo_，存放的就是通訊流封包。

而使用 **pkt_get()**，首先會取得 ether header，如下所示：

```
eh = (struct ether_header *)packet->pkt_get();
```

接著用 **macaddr_to_str** API 將 eh 中的 source mac address，跟 dest mac address 轉成 str 的 API。如下所示：

```
macaddr_to_str(eh->ether_shost, src_mac_str);
macaddr_to_str(eh->ether_dhost, dst_mac_str);
```

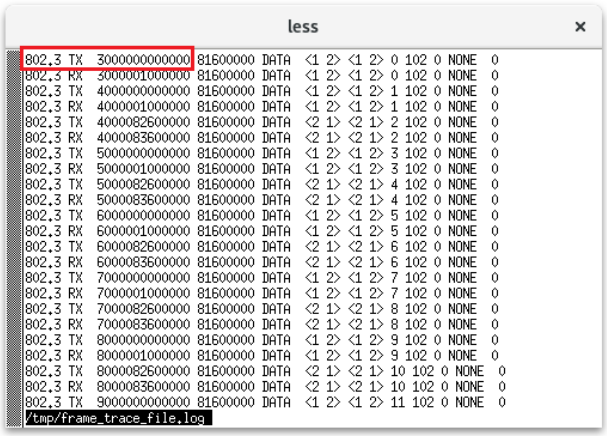
若要取得 IP HEADER，則是使用 **pkt_sget()**，如下所示：

```
iph = (struct ip *)packet->pkt_sget();
```

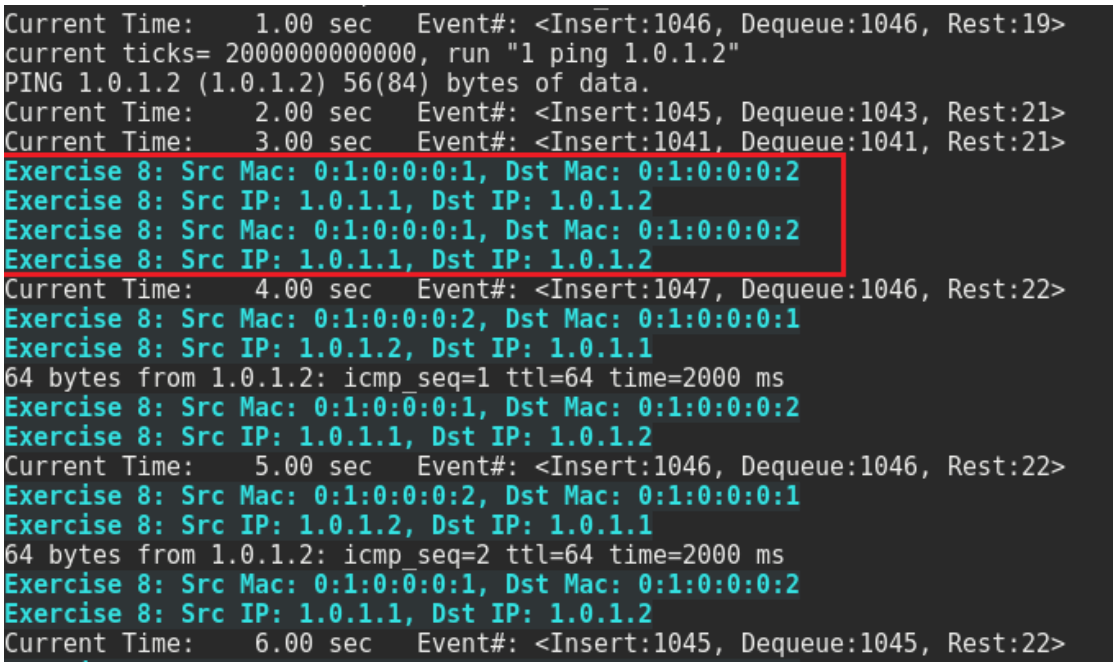
接著用 **ipv4addr_to_str** API 將 IP HEADER 中的 ip source address 以及 ip dest address 轉成 str，如下所示：

```
ipv4addr_to_str(iph->ip_src, src_ip_str);
ipv4addr_to_str(iph->ip_dst, dst_ip_str);
```

完成修改後，將原始碼 make 以及 make install 後執行模擬，模擬結果如下所示，可以看到原本第二秒就要送出的封包，在第三秒才有 log，達成了封包延遲的效果。



接著看一下 estinetss 的視窗，剛才使用 ipv4addr_to_string()以及 macaddr_to_str()所列印出來的資訊如下圖所示，開發者可以印此資訊來 debug 或是查看相關資訊。



■ 練習 8-2：pkt_addinfo()、pkt_getinfo()與 pkt_delinfo()

有時候在模組之間處理資料時，會希望封包有一些攜帶額外的資訊，但封包通常有特定的格式。因此在模擬器中的封包結構，每個封包都有一個額外的 buffer，可以存放一些額外的資訊，稱為 PT_INFO。可以使用 **pkt_addinfo()**、**pkt_getinfo()**、**pkt_delinfo()**等 API，讓封包可以攜帶額外的資訊。

user_module01.h 修改部分，如下紅字所示：

```
#ifndef __user_module_01_h__
#define __user_module_01_h__

#include <event.h>
#include <object.h>

class UserModule01 : public NslObject {

 private:
      int          Number;
      char         *String;

 public:
```

```
        UserModule01(u_int32_t type, u_int32_t id, struct plist* pl, const char *name);
        ~UserModule01();

        int             init();
        int             command(int argc, const char *argv[]);
        int             recv(ePacket_ *pkt);
        int             send(ePacket_ *pkt);
        void            pkt_delay(ePacket_ *pkt);
};

#endif    /* __user_module_01_h__ */
```

修改 user_module01.cc，延用上一練習紅色修改的部分，並加上藍色的修改如下所示：

```
#include <stdlib.h>
#include <estinet_api.h>
#include <module/user-defined/user_module_01.h>
#include <packet.h>
#include <ethernet.h>
#include <ip.h>

MODULE_GENERATOR(UserModule01);

UserModule01::UserModule01(u_int32_t type, u_int32_t id, struct plist* pl, const char *name)
        : NslObject(type, id, pl, name) {
}

UserModule01::~UserModule01(){}

int UserModule01::init() {
        return(NslObject::init());
}

int UserModule01::command(int argc, const char *argv[]) {
        return(NslObject::command(argc, argv));
}

int UserModule01::recv(ePacket_ *pkt) {
```

```
        return(NslObject::recv(pkt));
}

int UserModule01::send(ePacket_ *pkt) {
        u_int64_t    current_time_in_tick;
        u_int64_t    delay_time_in_tick;
        BASE_OBJTYPE(mem_func);

        current_time_in_tick = GetCurrentTime();
        SEC_TO_TICK(delay_time_in_tick, 1);
        mem_func = POINTER_TO_MEMBER(UserModule01, pkt_delay);
        setObjEvent(pkt, current_time_in_tick + delay_time_in_tick,
                0, this, mem_func, (void *)pkt->DataInfo_);
        return(1);

}

void UserModule01::pkt_delay(ePacket_ *pkt) {

        Packet                  *packet;
        struct ether_header     *eh;
        char                    src_mac_str[18];
        char                    dst_mac_str[18];

        struct ip               *iph;
        char                    src_ip_str[16];
        char                    dst_ip_str[16];

        if(pkt != NULL && pkt->DataInfo_ != NULL) {
                packet = (Packet *)pkt->DataInfo_;
                eh = (struct ether_header *)packet->pkt_get();

                macaddr_to_str(eh->ether_shost, src_mac_str);
                macaddr_to_str(eh->ether_dhost, dst_mac_str);

                printf("\e[1;36;40mExercise 8-2: Src Mac: %s, Dst Mac: %s\e[m\n",
                        src_mac_str, dst_mac_str);
```

```
            iph = (struct ip *)packet->pkt_sget();
            if(iph != NULL) {
                ipv4addr_to_str(iph->ip_src, src_ip_str);
                ipv4addr_to_str(iph->ip_dst, dst_ip_str);
                printf("\e[1;36;40mExercise 8-2: Src IP: %s, Dst IP: %s\e[m\n",
                src_ip_str, dst_ip_str);
            }
        }

        int NodeColor=random()%3;
        packet->pkt_addinfo("NodeColor", (char *)&NodeColor, sizeof(NodeColor));
        NslObject::send(pkt);
}
```

在這個範例中，User_module01 在傳送封包到下一個模組前，在每一個封包上加上一個 info，是 node 的顏色，有 3 種 random 值。

- pkt_addinfo 用法：

  第一個參數是此 info 的識別值(使用者定義，最多 15 個字元)，第二個參數則是一個 c++變數，第三個參數則是此 c++變數的大小。

在 phy.cc 修改部分如下藍字所示：

```
int phy::send(ePacket_ *pkt) {

        struct con_list             *cl;
        Packet                      *p;
        struct phyInfo              *phyinfo;

        assert(pkt&&(p=(Packet *)pkt->DataInfo_));
        if ( LinkFailFlag > 0 ) {
            freePacket(pkt);
            return(1);
        }

        int *nodecolor = (int *)p->pkt_getinfo("NodeColor");
        if(*nodecolor == 1)
```

```
            printf("\e[1;33;42mExercise 8-2: Node Color : %d\e[m\n", *nodecolor);
        else if(*nodecolor == 2)
            printf("\033[1;35;45mExercise 8-2: Node Color : %d\033[m\n", *nodecolor);
        else
            printf("\e[1;31;41mExercise 8-2: Node Color : %d\e[m\n", *nodecolor);
        p->pkt_delinfo("NodeColor");
```

在 phy 模組中，則用 pkt_getinfo()，取得此 NodeColor 資訊，並針對每一個值印出不同的 ascii 顏色。

處理後，使用 pkt_delinfo 將這個 NodeColor 資訊刪除。

- pkt_getinfo()與 pkt_delinfo()用法：

  pkt_getinfo 帶入 info 的識別字會回傳此識別字的數值；而 pkt_delinfo 帶入 info 的識別字會刪除此筆 info 的資訊。

執行結果：

```
Current Time:    1.00 sec   Event#: <Insert:1046, Dequeue:1046, Rest:19>
current ticks= 2000000000000, run "1 ping 1.0.1.2"
PING 1.0.1.2 (1.0.1.2) 56(84) bytes of data.
Current Time:    2.00 sec   Event#: <Insert:1045, Dequeue:1043, Rest:21>
Current Time:    3.00 sec   Event#: <Insert:1041, Dequeue:1041, Rest:21>
Exercise 8-2: Src Mac: 0:1:0:0:0:1, Dst Mac: 0:1:0:0:0:2
Exercise 8-2: Src IP: 1.0.1.1, Dst IP: 1.0.1.2
Exercise 8-2: Node Color : 0
Exercise 8-2: Src Mac: 0:1:0:0:0:1, Dst Mac: 0:1:0:0:0:2
Exercise 8-2: Src IP: 1.0.1.1, Dst IP: 1.0.1.2
Exercise 8-2: Node Color : 0
Current Time:    4.00 sec   Event#: <Insert:1047, Dequeue:1046, Rest:22>
Exercise 8-2: Src Mac: 0:1:0:0:0:2, Dst Mac: 0:1:0:0:0:1
Exercise 8-2: Src IP: 1.0.1.2, Dst IP: 1.0.1.1
Exercise 8-2: Node Color : 2
64 bytes from 1.0.1.2: icmp_seq=1 ttl=64 time=2000 ms
Exercise 8-2: Src Mac: 0:1:0:0:0:1, Dst Mac: 0:1:0:0:0:2
Exercise 8-2: Src IP: 1.0.1.1, Dst IP: 1.0.1.2
Exercise 8-2: Node Color : 1
Current Time:    5.00 sec   Event#: <Insert:1046, Dequeue:1046, Rest:22>
Exercise 8-2: Src Mac: 0:1:0:0:0:2, Dst Mac: 0:1:0:0:0:1
Exercise 8-2: Src IP: 1.0.1.2, Dst IP: 1.0.1.1
Exercise 8-2: Node Color : 0
64 bytes from 1.0.1.2: icmp_seq=2 ttl=64 time=2000 ms
Exercise 8-2: Src Mac: 0:1:0:0:0:1, Dst Mac: 0:1:0:0:0:2
Exercise 8-2: Src IP: 1.0.1.1, Dst IP: 1.0.1.2
Exercise 8-2: Node Color : 0
Current Time:    6.00 sec   Event#: <Insert:1045, Dequeue:1045, Rest:22>
Exercise 8-2: Src Mac: 0:1:0:0:0:2, Dst Mac: 0:1:0:0:0:1
Exercise 8-2: Src IP: 1.0.1.2, Dst IP: 1.0.1.1
Exercise 8-2: Node Color : 2
64 bytes from 1.0.1.2: icmp_seq=3 ttl=64 time=2000 ms
Exercise 8-2: Src Mac: 0:1:0:0:0:1, Dst Mac: 0:1:0:0:0:2
Exercise 8-2: Src IP: 1.0.1.1, Dst IP: 1.0.1.2
Exercise 8-2: Node Color : 2
Current Time:    7.00 sec   Event#: <Insert:1047, Dequeue:1047, Rest:22>
```

第九章：其它 API 練習

本章重點：
(1)、介紹 GetNodeLoc、GetTotalNumOfNodes、GetConFilePathAndName、getConnectNode 等 API

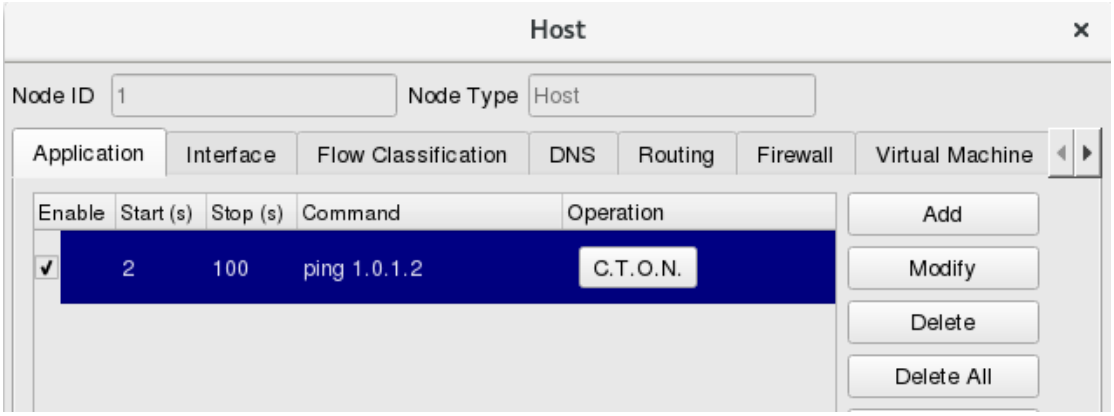
下載練習：

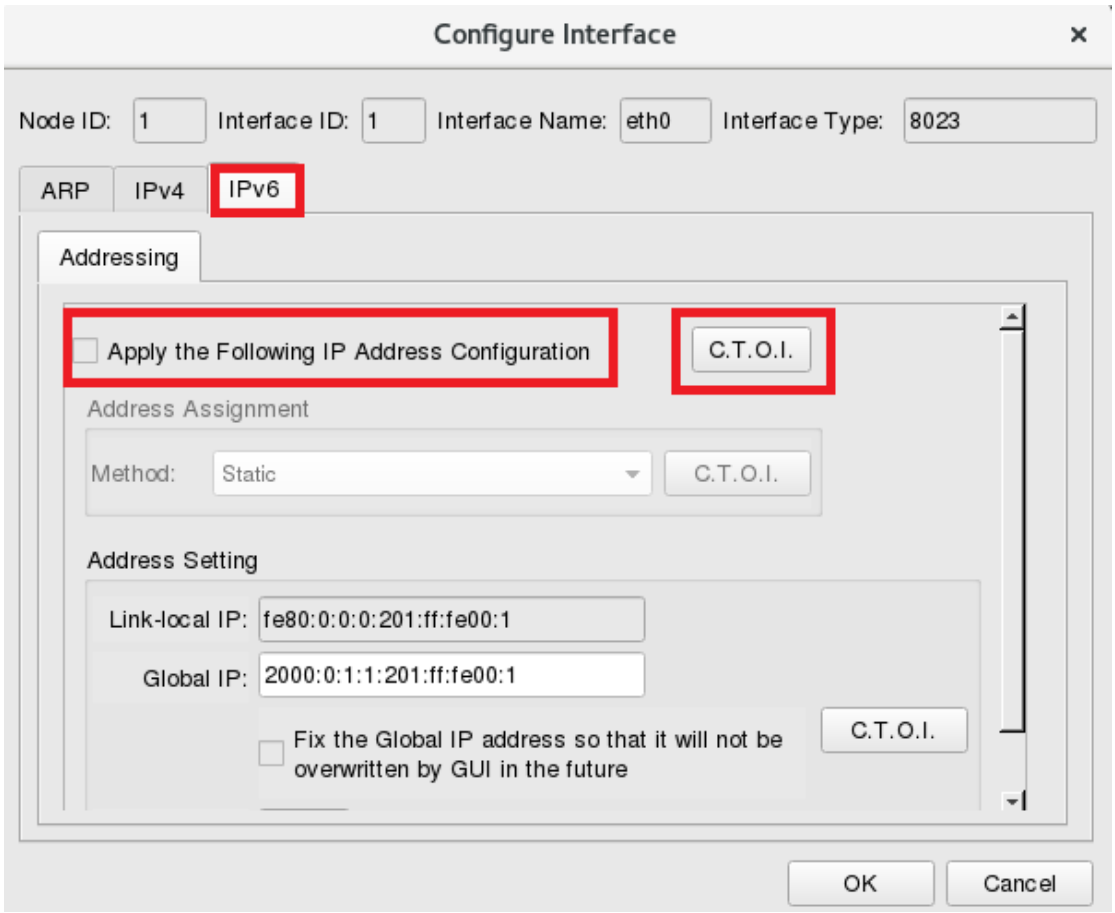

Chapter9.tar.bz2

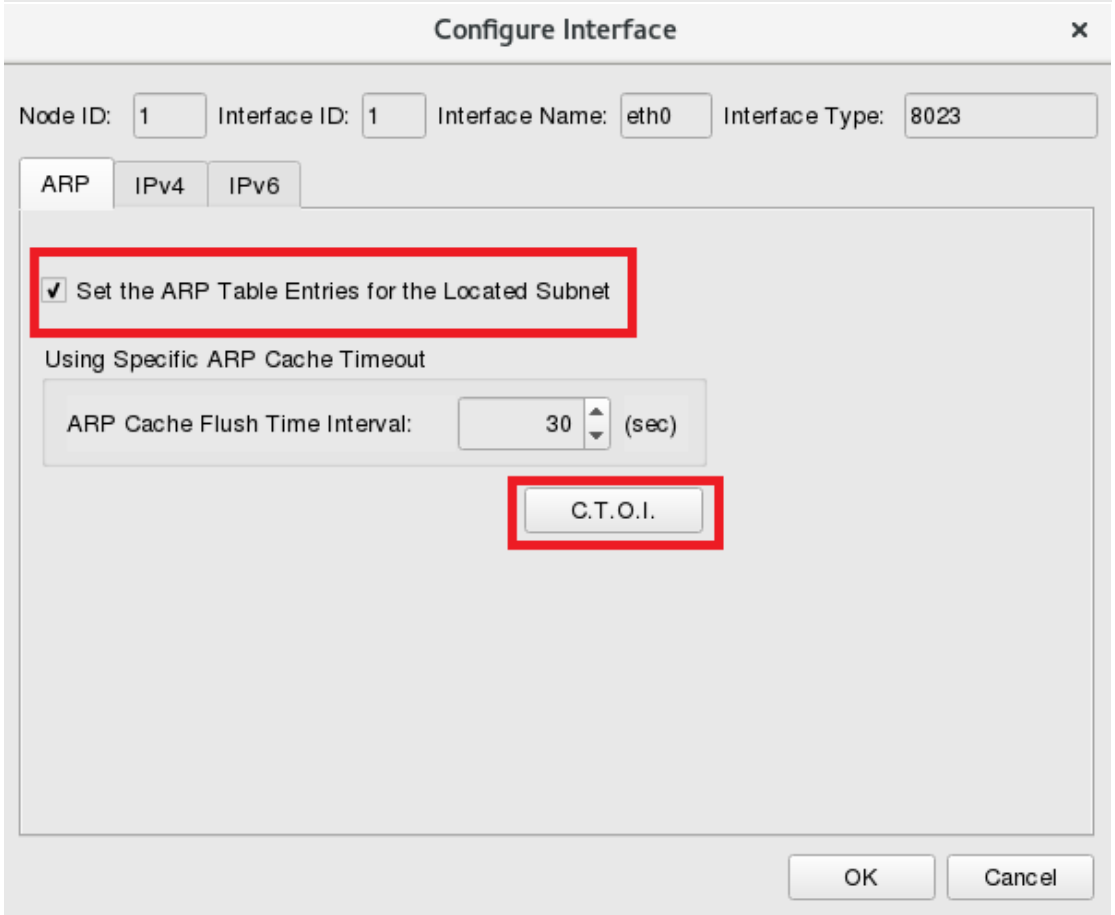
本章中介紹並使用一些常用的 API。

■ 練習 9-1

同樣使用第一章的範例拓撲(user_module01.xtpl)，並加入通訊流如下：
首先使用 Host1 ping Host2，在 Host1 中加入 ping 1.0.1.2。在 Host1 與 Host2 上都關閉 ARP 與 IPv6 介面。




Configure Interface
Node ID: 1
Interface ID: 1
Interface Name: eth0
Interface Type: 8023
ARP
IPv4
IPv6
Addressing
Apply the Following IP Address Configuration
C.T.O.I.
Address Assignment
Method: Static
C.T.O.I.
Address Setting
Link-local IP: fe80:0:0:0:201:ff:fe00:1
Global IP: 2000:0:1:1:201:ff:fe00:1
Fix the Global IP address so that it will not be overwritten by GUI in the future
C.T.O.I.
OK
Cancel


Configure Interface
Node ID: 1
Interface ID: 1
Interface Name: eth0
Interface Type: 8023
ARP
IPv4
IPv6
Set the ARP Table Entries for the Located Subnet
Using Specific ARP Cache Timeout
ARP Cache Flush Time Interval: 30 (sec)
C.T.O.I.
OK
Cancel

修改 user_module01.h 如下紅字所示：

```
#ifndef __user_module_01_h__
#define __user_module_01_h__

#include <event.h>
#include <object.h>

class UserModule01 : public NslObject {

 private:
        int             Number;
        char            *String;

 public:

        UserModule01(u_int32_t type, u_int32_t id, struct plist* pl, const char *name);
        ~UserModule01();

        int             init();
        int             command(int argc, const char *argv[]);
        int             recv(ePacket_ *pkt);
        int             send(ePacket_ *pkt);
        void            pkt_delay(ePacket_ *pkt);
};

#endif    /* __user_module_01_h__ */
```

使用第八章的 8-1 的範例，如下面紅字所示。並加上這章節的藍字部分。

user_module01.cc

```
#include <stdlib.h>
#include <estinet_api.h>
#include <module/user-defined/user_module_01.h>
#include <packet.h>
#include <ethernet.h>
#include <ip.h>

MODULE_GENERATOR(UserModule01);
```

```
UserModule01::UserModule01(u_int32_t type, u_int32_t id, struct plist* pl, const char *name)
        : NslObject(type, id, pl, name) {
        vBind_int("myNumber", &Number);
        vBind_char_str("myString", &String);
        REG_VAR("shared-number", &Number);
}

UserModule01::~UserModule01(){}

int UserModule01::init() {
        return(NslObject::init());
}

int UserModule01::command(int argc, const char *argv[]) {
        return(NslObject::command(argc, argv));
}

int UserModule01::recv(ePacket_ *pkt) {
        return(NslObject::recv(pkt));
}

int UserModule01::send(ePacket_ *pkt) {
        u_int64_t    current_time_in_tick;
        u_int64_t    delay_time_in_tick;
        BASE_OBJTYPE(mem_func);

        current_time_in_tick = GetCurrentTime();
        SEC_TO_TICK(delay_time_in_tick, 1);
        mem_func = POINTER_TO_MEMBER(UserModule01, pkt_delay);
        setObjEvent(pkt, current_time_in_tick + delay_time_in_tick,
                0, this, mem_func, (void *)pkt->DataInfo_);
        return(1);
}

void UserModule01::pkt_delay(ePacket_ *pkt) {

        Packet                  *packet;
        struct ether_header     *eh;
```

```
        char                src_mac_str[18];
        char                dst_mac_str[18];

        struct ip           *iph;
        char                src_ip_str[16];
        char                dst_ip_str[16];

        if(pkt != NULL && pkt->DataInfo_ != NULL) {
                packet = (Packet *)pkt->DataInfo_;
                eh = (struct ether_header *)packet->pkt_get();

                macaddr_to_str(eh->ether_shost, src_mac_str);
                macaddr_to_str(eh->ether_dhost, dst_mac_str);

                printf("\e[1;36;40mExercise 9-1: Src Mac: %s, Dst Mac: %s\e[m\n",
                        src_mac_str, dst_mac_str);

                iph = (struct ip *)packet->pkt_sget();
                if(iph != NULL) {
                        ipv4addr_to_str(iph->ip_src, src_ip_str);
                        ipv4addr_to_str(iph->ip_dst, dst_ip_str);
                        printf("\e[1;36;40mExercise 9-1: Src IP: %s, Dst IP: %s\e[m\n",
                        src_ip_str, dst_ip_str);

                }
        }

        double x, y, z;
        GetNodeLoc(get_nid(), x, y, z);
        printf("\e[1;32;40mExercise 9-1: GetTotalNumOfNodes()=%d\e[m\n",
GetTotalNumOfNodes());
        printf("\e[1;32;40mExercise 9-1: GetConfigFilePathAndName()=%s\e[m\n",
GetConfigFilePathAndName());
        printf("\e[1;32;40mExercise 9-1: GetNodeLoc()=%f, %f, %f\e[m\n", x, y, z);
        printf("\e[1;32;40mExercise 9-1: getConnectNode()=%d\e[m\n", getConnectNode(get_nid(),
get_ifid()));
        printf("\e[1;32;40mExercise 9-1: GetPacketLength=%d\e[m\n", packet->pkt_getlen());
        NslObject::send(pkt);
```

```
}
```

紅字部分說明請參考第八章。

藍字部分則使用了數個 API，說明如下：

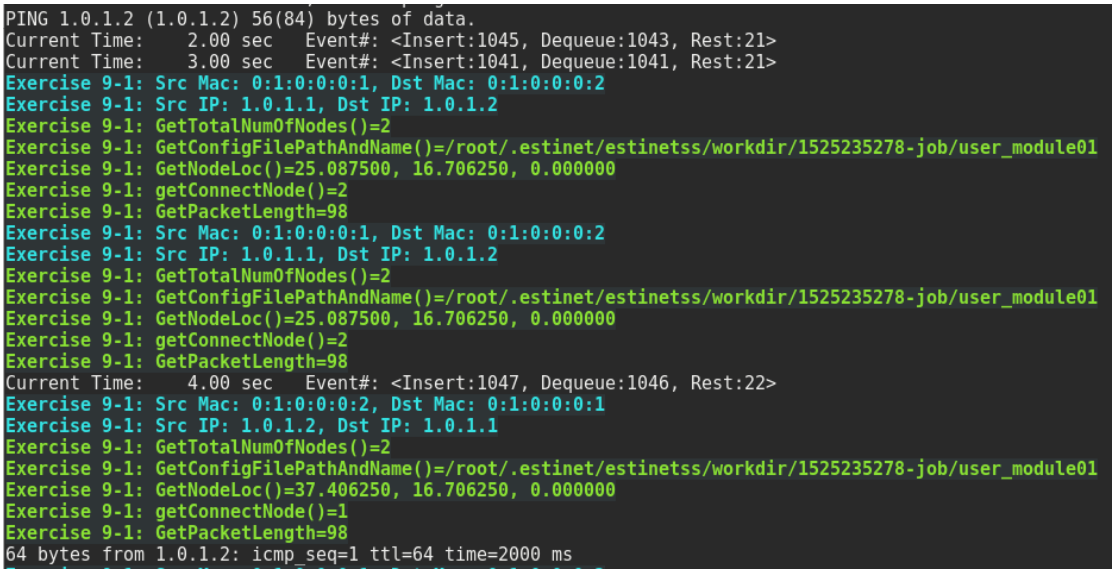
GetNodeLoc(nid, x, y, z)這個 API 可以得到 node 的位置。第一個參數是帶入自己的 node id。而回傳的資訊將會存在參數 x,y,z。

而 GetTotalNumOfNodes 這個 API 則回傳模擬環境中所有節點的數量。

GetConfigFilePathAndName 這個 API 回傳目前設定檔的檔案路徑跟這個拓撲名稱。

getConnectNode 這個 API，則是可以得到目前與這個 node id 相連的 node id，第一個參數要帶入 node id，第二個參數則是實體連線編號，使用 get_ifid()得知即可。

執行結果如下圖所示：



附錄

附錄 A、MDF HeaderSection 的細節

The first table collects the relevant variables and their meanings. The second table lists the set of possible values for each variable.

Field Name	Meaning
ModuleName	The name of this module
GroupName	The name of the group this module belongs to. And GUI will list GroupName in Node Editor to classify mdf file by GroupName.
Introduction	A short description or comment about this module
Parameter	A start-time parameter variable. The GUI program reads this part to know what parameters will be used at start-time. With this information, it will export these start-time parameters in the generated .if_and_medium_conffile.

TABLE THE MEANINGS OF THE VARIABLES USED IN THE HEADERSECTION

Field Name	Possible Values
ModuleName	Any user-specified string
GroupName	80211p, classification, interface, mac8023, mnode, phy, sdn_wifi_infra, traffic_control, vehicular_network, ap, hub, mac80211, mifx, openflow, pktgen, teleportal, user_defined, wphy (A user can create a new module group)
Introduction	Any user specified comment description string
Parameter	The format of a parameter statement is explained as follows: **Parameter Name Value Attribute** The possible attributes are listed below: “**local,**” “**gui_autogen,**” “**gui_autoassign,**” and “**local_only**”. “**local**” means that this parameter is used only in this module and if its value is updated, it will not be copied to other modules. Unless press "C.T.O.N." button, it can be copied to all modules on all nodes with the same node type. ”**gui_autogen**” means that the value of this parameter will be

	automatically generated by the GUI program. However, a user can **not** replace/change the auto-generated value with his (her) value. If a user press "C.T.O.N." button, the auto-generated value will be **not** copied to any module. Normally, a possible value of an **gui_autogen** parameter is a formula consisting of the three predefined variables: **$CASE$, NID, and PID.** **$CASE$** represents the main file name of a simulation case's topology file. It will be replaced by the main file name when this variable is accessed. For example, if a simulation case's topology file is saved with the filename "test.xtpl", **$CASE$** will be replaced by "test." **NID** represents the ID of the node to which this module is attached. Analogously, **PID** represents the ID of the interface to which this module is attached. "**gui_autoassign**" is similar to "**gui_autogen**" However, a user can **not** replace/change its value. No matter how a user replaces/changes the auto-generated value with his (her) desired one, the final value is still determined by a pre-defined formula. ex. ip address, mac address, interface id...etc.. "**local_only**" means that the value of this parameter can be replace/change the value with his (her) value. If a user press "C.T.O.N." button, the value will be **not** copied to any module. Because the parameter must be set respectively.

TABLE THE POSSIBLE VALUES FOR THE VARIABLES USED IN THE HEADERSECTION

附錄 B、MDF InitVariable Section 的細節

Normally, a user should specify the caption and the size of the diabox. The key word "**Caption**" indicates the caption of the dialog box, and "**FrameSize width height**" indicates the size of the dialog box. For example,

```
Caption         "Parameters Setting"
FrameSize       340 80
```

These statements will generate a dialog box of 340x80 pixels with a caption of "Parameters Setting." After specifying the caption and the size of the dialog box, a user can arrange the layout inside the dialog box. A dialog box would contain a number of

GUI objects, such as an OK button, a Cancel button, a textline, etc. Each GUI object corresponds to a description block in “InitVariableSection” and always starts with “Begin” and ends with “End.” The following shows an example:

```
Begin BUTTON        b_ok
    Caption         "OK"
    Scale           270 12 60 30
    ActiveOn        MODE_EDIT
    Enabled         TRUE
    Action          ok
    Comment         "OK Button"
End
```

The description blocks for different objects share several common and basic attributes. For example, the caption and scale commands are used commonly. A “BUTTON”-like object is an example of an object consisting of only basic attributes. Let’s take the simple “BUTTON” object as an example. More specific attributes will be discussed later.

For a “BUTTON” object, the keyword “BUTTON” follows the keyword “Begin” and it is followed by the object name “b_ok”. The following table lists its attributes:

Attribute name	Possible values	Comment
Caption	User-specified	The caption of this object
Scale	User-specified	The four numbers represent (x, y, width, height).
ActiveOn	MODE_EDIT, MODE_SIMULATION, ALL_MODE	An option to specify in which mode this object should is active. The “MODE_EDIT” stand for the object is enabled at Edit Mode. The “MODE_SIMULATION” stand for the object is enabled at GUI G Mode. ALL_MODE indicates that the object will be activated whatever the Mode is.
Enabled	TRUE, FALSE	If an object is not enabled, it will be dimly displayed. That is, a user cannot operate this object.
Action	ok , cancel	An attribute used by button-like objects, such as the OK button and cancel

		buttons to indicate which action it should perform when a user presses it.
Comment	User-specified	Comment for this object

TABLE THE BASIC ATTRIBUTES USED TO DESCRIBE AN OBJECT

a. LABEL

"LABEL" is used to display some comment in a dialog box. The attributes of a LABEL object are the same as those of a "BUTTON" object. An example is following below:

```
Begin LABEL         l_ums
      Caption       "(ms)"
      Scale         220 24 35 35
      ActiveOn      MODE_EDIT
      Enabled       FALSE
End
```

b. GROUP

GROUP is used to organize related objects together. It can contain a number of objects that are related to an area. Like other objects, it has four basic attributes "**Caption**," "**Scale**," "**ActiveOn**," and "**Enabled**" to define the caption, the size of its area, the active mode, and the enabled/disabled conditions. An example is following below. The group has four objects include a Radiobox, two textline, a lable.

```
Begin Group         g_radio
      Caption       "Mode"
      Scale         10 15 260 135
      ActiveOn      MODE_EDIT
      Enabled       TRUE

      Begin RADIOBOX      myString
            Option        "op1"
            Enable        myNumber
            Enable        lable1
            OptValue      "string1"
            VSpace        5
            EndOption

            Option        "op2"
```

```
            Disable         myNumber
            Disable         lable1
            OptValue        "string2"
            VSpace          40
            EndOption

            Type            STRING
            Comment         "radiobox test"
        End

        Begin TEXTLINE      myNumber
            Caption         "input Number"
            Scale           35 35 180 35
            ActiveOn        MODE_EDIT
            Enabled         FALSE
            Type            INT
            Comment         "for test"
        End

        Begin LABEL         lable1
            Caption         "(INT)"
            Scale           220 35 35 35
            ActiveOn        MODE_EDIT
            Enabled         FALSE
        End
    End
```

c. RADIOBOX/CHECKBOX

In RADIOBOX/CHECKBOX, there are some new attributes. (Note: Outside of a **radiobox** must be a group object) Let's take the following example to explain:

```
    Begin Group         g_radio
        Caption         "Mode"
        Scale           10 15 260 135
        ActiveOn        MODE_EDIT
        Enabled         TRUE

        Begin RADIOBOX  myString
```

```
            Option          "op1"
            Enable          myNumber
            Enable          lable1
            OptValue        "string1"
            VSpace          5
            EndOption

            Option          "op2"
            Disable         myNumber
            Disable         lable1
            OptValue        "string2"
            VSpace          40
            EndOption

            Type            STRING
            Comment         "radiobox test"
        End

    Begin TEXTLINE      myNumber
            Caption     "input Number"
            Scale       35 35 180 35
            ActiveOn    MODE_EDIT
            Enabled     FALSE
            Type        INT
            Comment     "for test"
    End

    Begin LABEL         lable1
            Caption     "(INT)"
            Scale       220 35 35 35
            ActiveOn    MODE_EDIT
            Enabled     FALSE
    End
End
```

It is a RADIOBOX block whose name is "***myString***." The two option blocks follow, each of which starts with the "**Option**" keyword and ends with the "**EndOption**" keyword. The string following the "Option" keyword specifies the string that should be shown in

the dialog box for this option. The "**OptValue**" specifies the value that will be assigned to the radiobox option variable "***myString***" if this option is selected. The "Enable" and "Disable" statements inside an "Option" block specify that, when a user selects this option, the variable objects following these statements should be enabled or disabled (When an object is enabled, its input field is enabled in the parameter dialog box, otherwise, its input field is disabled). The term "**VSpace**" is used to specify the vertical height of the area used for outside group's y location(only using for Radiobox). The term "**Comment**" is used to specify comment for this object.

```
Begin CHECKBOX      check1
      Caption       "Set My Number"
      Scale         10 50 180 20
      ActiveOn      MODE_EDIT
      Enabled       TRUE

      Option        "TRUE"
      OptValue      "on"
      Enable        myNumber
      EndOption

      Option        "FALSE"
      OptValue      "off"
      Disable       myNumber
      EndOption
      Comment       ""
End
```

The following is a checkbox block whose name is "*check1*." Like other objects, it has four basic attributes "**Caption**," "**Scale**," "**ActiveOn**," and "**Enabled**" to define the caption, the size of its area, the active mode, and the enabled/disabled conditions. And then, the two option blocks follow, each of which starts with the "**Option**" keyword and ends with the "**EndOption**" keyword. The string following the "Option" keyword specifies the string that should be shown in the dialog box for this option. The "**OptValue**" specifies the value that will be assigned to the checkbox option variable "*check1*" if this option is selected. The "Enable" and "Disable" statements inside an "Option" block specify that, when a user selects this option, the variable objects following these statements should be enabled or disabled (When an object is enabled, its input field is enabled in the parameter dialog box, otherwise, its input field is

disabled). The term “**Comment**” is used to specify comment for this object.

d. TEXTLINE

TEXTLINE provides a text field for inputting or outputting data. Like other objects, it has four basic attributes “**Caption**,” “**Scale**,” “**ActiveOn**,” and “**Enabled**” to define the caption, the size of its area, the active mode, and the enabled/disabled conditions. A module developer can indicate the type of the data to be read from a textline. The data will be interpreted as a value of the type indicated by the “TYPE” key word. But now we only support “**STRING**” for “TYPE”, other data type not yet. The term “**Comment**” is used to specify comment for this object. An example is following below.

```
Begin TEXTLINE      myNumber
      Caption       "My Number "
      Scale         10 70 220 30
      ActiveOn      MODE_EDIT
      Enabled       FALSE
      Type          INT
      Comment       "An Integer"
End
```

附錄 C、MDF 中的 Export Section 細節

“ExportSection” provides an area in a dialog box in which a user can get/set the current value of a variable at run-time. “**Caption**”, “**FrameSize**” are the two basic attributes for this section. If a module doesn’t have any variable that can be accessed during simulation, “Caption” should be set to “”, a null string, and “FrameSize” should be set to 0 0. Or the ExportSection does not be added.

```
ExportSection
      Caption       ""
      FrameSize     0 0
EndExportSection
```

In addition to the objects discussed above, there are two useful objects that are new in this section. They are the “**INTERACTIONVIEW**” and “**ACCESSBUTTON.**” The formats of these two objects are shown in the following examples:

```
Begin ACCESSBUTTON  ab_get_mystr
        Caption         "Get"
        Scale           215 50 70 25
        ActiveOn        MODE_SIMULATION
        Enabled         TRUE
        Action          GET
        ActionObj       "export-my-string"
        Reference       text_query_mystr
        Comment         "get"
End

Begin ACCESSBUTTON  ab_set_mynum
        Caption         "Set"
        Scale           290 15 70 25
        ActiveOn        MODE_SIMULATION
        Enabled         TRUE
        Action          SET
        ActionObj       "export-my-number"
        Reference       text_query_mynum
        Comment         "set"
End
```

For an "**ACCESSBUTTON**" object, it is used to get or set the value of a single-value run-time variable. There are three new attributes for "**ACCESSBUTTON**." They are "**Action,**" "**ActionObj,**" and "**Reference,**" respectively. The value of "Action" can be "**GET**" or "**SET**" to indicate when a user presses this button which operation should be performed. "**ActionObj**" indicates the name of the object that the GET/SET operation should operate on in the simulation engine. Finally, "**Reference**" points to the name of the GUI object (e.g., a TEXTLINE object) in which the retrieved value should be displayed. For example, the max queue length of a Interface module may be gotten and displayed at a TEXTLINE GUI object named "t_mq."

```
Begin INTERACTIONVIEW     iv_get_all
    Caption               "Get All Var"
    Scale                 10 100 200 30
    ActiveOn              MODE_SIMULATION
    Enabled               TRUE
    Action                GET
    ActionObj             "export-all-data"
    Fields                "My String" "My Number"
    Comment               "All Data"
End
```

For an "**INTERACTIONVIEW**" object, it is used to display the content of a multi-column table at run-time. Normally, it is used to get a switch table, an ARP table, or an AP's association table. Besides "**Action**" and "**ActionObj**," there is a new attribute called "**Fields**" to specify the names of the fields (columns) of the table. Several quoted strings, each of which represents the name of a field, follow the "**Fields**" attribute.

附錄 D、模擬器的分散式架構：

EstiNet uses a distributed architecture to support remote simulations and concurrent simulations. The estinetjd is used to do this task. It should be executed and remain alive all the time to manage multiple simulation machines. On every simulation machine, the estinetss needs to be executed and remain alive to let the estinetjd know whether currently this machine is busy running a simulation case or not. The following figure depicts the distributed architecture of EstiNet.

For example, the estinetjd in the simulation service center can accept simulation jobs from the whole world. When a user submits a simulation job to the estinetjd, the estinetjd selects an available simulation machine to service the job. If there is no available simulation machine, the job will be put into the job queue of the estinetjd. Every simulation machine always has a running estinetss to communicate with the GUI program and the estinetjd. The estinetss will notify the estinetjd whether the simulation machine managed by itself is available or not. When the estinetss receives a simulation job from the estinetjd, it forks (executes) a simulation engine process to simulate the specified network and protocols. When the simulation engine process is running, the estinetss will communicate with the estinetjd and the GUI program. For example, periodically the simulation engine process will send the current virtual time

of the simulation network to the estinetss. Then the estinetss will relay the information to the GUI program. This enables the GUI user to know the progress of the simulation. During a simulation, the user can also on-line set or get a protocol module's value (e.g. to query a switch's switch table). Message exchanges happening between the simulation engine process and the GUI program are all done via the estinetss.